НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ
КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ СИСТЕМНОГО АНАЛІЗУ

«До захисту допущено»
Завідувач кафедри
_____ О. Л. Тимощук
“___”_____2019 р.

Дипломна робота

на здобуття ступеня бакалавра

з напряму підготовки 6.050101 “Комп’ютерні науки”

на тему: “Система розпізнавання української мови”

Виконала:
студентка 4 курсу, групи КА-55
Туголукова Євгенія Валеріївна

Керівник:
доцент кафедри ММСА, к.т.н.
Дідковська М.В. _____

Консультант з економічного розділу:
доцент, к.е.н. Шевчук О. А. _____

Консультант з нормоконтролю:
доцент, к.т.н. Коваленко А. Є. _____

Рецензент:
доцент, к.т.н. Заболотня Т. М. _____

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.
Студент _____

Київ – 2019

Національний технічний університет України
«Київський політехнічний інститут
імені Ігоря Сікорського»

Інститут (факультет) ННК “Інститут прикладного системного аналізу”
Кафедра Математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки 6.050101 Комп’ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О. Л. Тимощук
«___»_____ 2019 р.

ЗАВДАННЯ
на дипломну роботу студенту
Туголуковій Євгенії Валеріївні

1. Тема роботи Система розпізнавання української мови,

керівник роботи Дідковська Марина Віталіївна, к.т.н., доц.,
затверджені наказом по університету від «___»_____ 20__ р. №_____

2. Термін подання студентом роботи _____

3. Вихідні дані до роботи : акустична модель, мовна модель, система конверсії слів у фонеми, словник розпізнавання.

4. Зміст роботи : дослідження та особливості предметної області систем розпізнавання, процес розпізнавання мовлення.

5. Перелік завдань, які потрібно розробити:

(1) дослідити актуальність задачі розпізнавання мовлення сьогодні;
(2) виконати огляд сучасних моделей розпізнавання мови;
(3) обрати та описати методи розпізнавання на основі Прихованих Марківських моделей;
(4) виконати обчислювальні експерименти стосовно якості розпізнавання;
(5) створити програмний продукт, виконати аналіз та порівняння отриманих результатів;

6. Консультанти розділів роботи*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Шевчук О. А., доцент		

7. Дата видачі завдання 15 квітня 2019 р.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Затвердження теми БДР	15.04.2019-21.04.2019	Виконано
2.	Ознайомлення зі структурою БДР згідно з Положенням про державну атестацію студентів НТУУ «КПІ»	15.04.2019-28.04.2019	Виконано
3.	Ознайомлення з ДСТУ 3008-95 та стандарти ЄСПД	21.04.2019-28.04.2019	Виконано
4.	Проведення дослідження за темою БДР під керівництвом керівника	28.04.2019-05.05.2019	Виконано
5.	Завершення роботи над першим варіантом частини БДР	05.05.2019-12.05.2019	Виконано
6.	Проведення роботи над експериментальною частиною БДР	05.05.2019-19.05.2019	Виконано
7.	Проведення роботи над програмним продуктом	19.05.2019-26.05.2019	Виконано
8.	Оформлення БДР та аналіз отриманих результатів	19.05.2019-26.05.2019	Виконано

Студент _____ Є.В. Туголукова

Керівник роботи _____ М.В. Дідковська

* Консультантом не може бути зазначено керівника дипломної роботи.

РЕФЕРАТ

Дана дипломна робота містить 78 ст., 4 ч., 9 табл., 15 рис., 2 дод., 18 джерел.

РОЗПІЗНАВАННЯ МОВЛЕННЯ, MFCC-ВЕКТОРИ, ПРИХОВАНІ МАРКІВСЬКІ МОДЕЛІ, СИСТЕМА ГОЛОСОВОГО УПРАВЛІННЯ

Об’єкт дослідження – процес розпізнавання української мови.

Мета та цілі роботи – розглянути теоретичне підґрунтя процесу розпізнавання мовлення, провести огляд існуючих методів та засобів розпізнавання, розробка засобів програмного забезпечення для автоматичного розпізнавання української мови.

Методи дослідження – аналіз процесу розпізнавання мовлення, експеримент, результатом якого є програмний продукт, аналіз отриманих результатів.

Результатом роботи є система розпізнавання української мови, а саме система голосового локального управління комп’ютером.

Новизною роботи є створення способу розпізнавання української мови на основі Прихованих Марківських моделей та з використанням JSGF-граматик, а також розробка програмного забезпечення з використанням знайденого способу.

Результати даної роботи можна застосовувати для полегшення та пришвидшення роботи з операційною системою Mac OS за допомогою голосового управління. Також, створений модуль розпізнавання української мови є універсальним та може бути використаний у будь-якому додатку, де необхідно розпізнавати злитне мовлення.

ABSTRACT

This thesis contains 78 p., 4 sections, 9 tabl., 16 fig., 2 appendixes, 18 sources.

SPEECH RECOGNITION, MFCC-VECTORS, HIDDEN MARKOV MODELS, VOICE USER-INTERFACE SYSTEM

The object of this work is the recognition process of the Ukrainian language.

The purpose and aims of this work is to consider the theoretical basis of the speech recognition process, to conduct an overview of existing methods and means of recognition, development of software tools for automatic recognition of the Ukrainian language.

Research methods - analysis of speech recognition process, experiment, the result of which is a software product, analysis of the results.

The result of the work is the recognition system of the Ukrainian language, namely the system of voice local control of the computer.

The novelty of the work is to create a way to recognize the Ukrainian language based on Hidden Markov models and using JSGF-grammar, as well as software development using the found method.

The results of this work can be used to facilitate and speed up the operation of the Mac OS. Also, the Ukrainian language recognition module is universal and can be used in any application where it is necessary to recognize interlaced speech.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

ASR – Authomatic Speech Recognition

ЕОМ – електронно-обчислювальна машина

ОС – операційна система

G2P – Grapheme-to-Phoneme

MFCC – Mel-frequency cepstrum coefficients

DTW – Dynamic Time Wrapping

ПММ – Приховані Марківські моделі

РНМ – рекурентні нейронні мережі

URL – Uniform Resource Locator

ДКЧП – довга короткочасна пам’ять

JSGF – JavaScript Grammar File

WER – World Error Rate

ВСТУП

Розпізнавання мовлення – це технологія, яка дозволяє комп’ютеру чи програмі ідентифікувати окремі слова або фрази, сказані людиною, та перетворити їх у текст. Вона включає в себе знання та дослідження у галузях інформатики, лінгвістики та електротехніки.

Системи розпізнавання мовлення поступово займають місце посередника між людиною та девайсом, і таким чином створюють альтернативу звичним засобам обміну інформацією. Поряд з програмним забезпеченням для диктування на персональних комп’ютерах, розвиваються більш прогресивні системи, такі як голосові асистенти, які, окрім виконання команд, імітують живий діалог та розв’язують прикладні задачі. Ці системи можуть бути реалізовані, як програма на комп’ютері чи смартфоні, так і як окремий девайс. Але для більшості з них необхідно обов’язково мати доступ в Інтернет. Таким чином, їх використання є обмеженим, а швидкість роботи напряму залежить від якості Інтернет-з’єднання. Ще потрібно сказати, що у більшості таких систем відсутня підтримка слов’янських мов. На це є свої причини.

Особливістю слов'янських мов, в тому числі і української, є висока флективність і відносно вільний порядок слів у фразі або реченні. Це призводить до швидкого зростання словника для розпізнавання (у 8-10 разів більше, ніж для тієї ж самої англійської лексики) і зменшення точності прогнозування мовної моделі. Тому застосування традиційних методів і алгоритмів до слов'янських мов виглядає досить неперспективно. Це є причиною удосконалення існуючих методів розпізнавання та пошуку нових.

Сучасні системи мовлення можуть розпізнавати ізольовані слова та чітку мову, як приклад, читання новин, з частотою помилок у 5%. Проте розпізнавання розмовної або злитної мови є менш надійним. Спонтанне розпізнавання мови в реалістичних умовах зв'язку (наприклад, фоновий шум)

є надзвичайно актуальною проблемою, рішення якої значно розширить застосування систем розпізнавання мовлення.

Тому об’єктом даної роботи є побудова системи розпізнавання української мови в реальному часі, яка буде працювати на сучасному персональному комп'ютері, причому локально, без доступу до Інтернет-мережі.

Перший розділ більш детально розкриває актуальність задачі, обґоворюється історичне підґрунтя для розвитку систем, сучасні підходи та проблеми розробок. У розділі 2 крок за кроком буде описано процес розпізнавання – від отримання звукової доріжки і до моменту запису тексту, який їй відповідає. Будуть розглянуті основні методи та засоби для розпізнавання мови та обрані найбільш підходящі для нашої задачі. Розділ 3 присвячений вибору архітектури нашої системи розпізнавання, опису її реалізації, та аналізу отриманих результатів. У четвертому розділі буде проведений функціонально-вартісний аналіз роботи.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність задачі розпізнавання української мови

Ще з моменту появи перших комп'ютерів вчені почали працювати над задачею автоматичного розпізнавання мови, оскільки робота з текстовим командним рядком на ЕОМ не завжди забезпечувала прийнятну швидкість і мала не дуже природний вигляд. Протягом багатьох років велись розробки, було створено безліч методів і комп'ютерних програм, для того, аби все ж таки вирішити проблему розпізнавання мови.

На сьогоднішній день у цій сфері ведеться активний розвиток, щороку великі компанії створюють нові та оновлюють діючі системи з розпізнаванням мовлення. В основному, такі системи підтримують англійську мову, але також можна додати іспанську, китайську, французьку та німецьку. Вибір мови напряму залежить від розвитку мовних технологій з економічної та політичної точки зору. Англійська мова є однією з найпоширеніших у світі, вона являється офіційною мовою більш ніж в 60 державах. Відповідно інвестиції, що вкладаються в розвиток технологій для розпізнавання цієї мови, досить швидко приносять дохід. Але на відміну від англійської, іншим мовам приділяється не так багато уваги, внаслідок чого розвиток їх технологій розпізнавання йде значно повільніше.

Тим часом, українською мовою спілкуються близько 50 млн. людей, за кількістю мовців вона є другою слов'янською мовою (після російської) та входить до тридцятки найпоширеніших мов світу. Незважаючи на це, діючих систем автоматичного розпізнавання українського спонтанного мовлення фактично не існує. На розвиток мовних технологій, крім економічних проблем, в першу чергу, впливають особливості української мови та складнощі, що виникають у процесі її обробки. Як основні, можна виділити: відсутність строгих граматичних конструкцій побудови речень, а також численні правила словотворення, представлення слів і розстановки наголосів, причому практично до всіх правил існують винятки.

1.1.1 Історія розвитку систем розпізнавання мови

Розробки в цій галузі почались ще понад півстоліття тому. Першим успішним прикладом даної технології стала система “Одрі”, створена у Bell Laboratories, яка могла розпізнавати цифри з точністю до 90% відсотків. З кожним десятиліттям технологія розпізнавання мови знаходила все більше застосувань, а словник розпізнаних слів поступово збільшувався. Більш ранні системи використовували лише звукові біти («фонеми») для розпізнавання. Однак, користувачам доводилося робити паузи і говорити повільно, аби машина справді могла розібрати вхідний звуковий потік. Згодом для покращення роботи до акустичного підходу вирішили додати лінгвістичний. Окрім того, щоб використовувати лише звуки, інженери почали впроваджувати алгоритми на основі мовних правил. Тобто, якщо б у системи виникли проблеми з розпізнаванням сказаного слова, вона б змогла зробити припущення, побудоване на семантичних, синтаксичних і тональних правилах.

Лише у 1997 році з’явився перший у світі розпізнавач безперервної мови названий Dragon Dictate. Він дозволяв не робити спеціальні паузи між словами та міг розуміти до 100 слів в хвилину.

Справжнім проривом для технології розпізнавання мовлення стало машинне навчання. З появою глибинного навчання, кількість помилок при розпізнаванні слів кардинально зменшилась. У 2010 році компанія Google поєднала алгоритми машинного навчання з потужними хмарними обчисленнями для обміну даними та випустила у світ продукт Google Voice Search. Даний застосунок використовував елементи персоналізації для побудови мовних шаблонів та збільшення точності розпізнавання. Згодом Google використав отримані дані при створенні власного голосового асистенту Google Assistant, який на сьогоднішній день встановлений більш ніж на 50% відсотках усіх смартфонів світу.

Рік згодом, компанія Apple представила свій голосовий асистент Siri з елементами штучного інтелекту, який став відомий своєю здатністю точно обробляти природні висловлювання та відповідно на них реагувати. Успіх Siri дав сильний поштовх технології розпізнавання мовлення. Такі компанії як Amazon, Microsoft почали розробляти власні продукти у цьому напрямі.

1.1.2 Приклади використання систем розпізнавання мови

На сьогоднішній день, системи розпізнавання мовлення використовуються практично на всіх девайсах та у багатьох життєво важливих галузях. Ми щоденно використовуємо голосове управління, команди та пошук в автомобілях, смартфонах та персональних комп'ютерах. Все популярнішими стають так звані системи "Розумний дім", у яких за допомогою голосових команд людина може управляти технікою та побутом в домі.

Системи розпізнавання також можуть бути корисними для вивчення мови. Вони є критично необхідними для людей з фізичними вадами, які мають проблеми з зором або не можуть писати та набирати текст.

У медичному секторі розпізнавання мовлення знайшло застосування для роботи з документацією. У військовій галузі ці системи використовують для голосового керування літальними апаратами, а в майбутньому планується таке ж саме застосування і в аерокосмічній промисловості. Якщо говорити про телефонію, то це автоматизація обробки вхідних і вихідних дзвінків, голосові системи самообслуговування для отримання довідкової інформації та консультацій, надання послуг та товарів, проведення опитувань, анкетування, збір інформації.

1.2 Існуючі підходи до вирішення задачі розпізнавання мови

1.2.1 Класифікація систем розпізнавання мовлення

Типи систем розпізнавання мовлення на основі висловлювань

Ізольована система розпізнавання слів

Розпізнає окремі висловлювання. Зручно використовувати у ситуаціях, коли користувачеві потрібно сказати лише одне слово або команду. Таку систему найпростіше реалізувати, оскільки нам дані межі слова, і слова, зазвичай, чітко виражені, що являє собою головну перевагу такого розпізнавання.

Зв'язані слова

Система зв'язаних слів подібна до ізольованих слів, але вона дозволяє окремим висловлюванням бути «зібраними разом» з мінімальним паузами між ними. Висловлювання – це вокалізація слова або слова, які представляють єдине значення для комп'ютера.

Неперервна мова

Система безперервного розпізнавання мови дозволяє користувачам говорити майже природно, поки комп'ютер визначає його зміст. В основному, це комп'ютерна диктування. При цьому найближчі слова виконуються разом без паузи або будь-якого іншого поділу між словами. Систему розпізнавання безперервної мови важко розробити.

Спонтанне мовлення

Система спонтанної розпізнавання мови розпізнає природну мову. Природно, що спонтанна мова приходить раптово через рот. Система ASR з спонтанною мовою здатні обробляти різноманітні природні функції, такі як слова, які виконуються разом. Спонтанна мова може включати помилкові вимоги, помилкові початки і не слова.

Типи систем розпізнавання мовлення за ознакою множини дикторів моделі спікера

Кожен спікер має особливий голос (тембр звучання), завдяки своїм унікальним фізіологічним та особистісним показникам. Системи розпізнавання мовлення за цією ознакою класифікуються за двома категоріями:

Дикторозалежні моделі

Такі системи розробляються під конкретних дикторів. Саме для цих людей вони будуть працювати максимально точно, в той час як з розпізнаванням інших голосів можуть виникати труднощі. Такі системи дешево та легко навчати і розвивати. Але водночас у них відсутня гнучкість, яка притаманна дикторонезалежним системам.

Дикторонезалежні моделі

Дані системи можуть розпізнавати будь-які голоси без попереднього навчання. Одним з базових застосувань таких моделей є системи інтерактивних голосових відповідей, які приймають вхідні дані від великої кількості користувачів. Недолік полягає в тому, що кількість слів у словнику обмежена. До того ж, розробка дикторонезалежних систем це складний процес, який дорого коштує. При цьому, точність такої системи на виході буде нижча за дикторозалежну.

Типи систем розпізнавання мовлення за об’ємом словника

Розмір словника системи розпізнавання мовлення напряму впливає на складність, обробку і швидкість розпізнавання системи. Таким чином, система на основі об’єму словника класифікується наступним чином:

- Малий словник – від 1 до 100 слів або фраз
- Середній словник – від 101 до 1000 слів або фраз
- Великий словник – від 1001 до 10000 слів або фраз
- Дуже великий словник – більше 10 000 слів або фраз

1.2.2 Аналіз існуючих платформ

Розглянемо декілька найпопулярніших систем розпізнавання мови.

Світовим лідером у створенні ПЗ для розпізнавання мови є Nuance Communications. Інструментарій Dragon Mobile SDK складається з компонентів клієнта і сервера, а також містить у собі різноманітні приклади коду і шаблони, документацію і фреймворк, що значно спрощує інтеграцію сервісів в додатку. Платформа Speech Kit створена для простого і швидкого додавання сервісів розпізнавання і синтезу мови в проекти і програми. За більшу частину операцій процесу обробки мови відповідає система серверів. На сервері повністю виконується розпізнавання та синтез мовлення.

Google Speech Recognition API – продукт компанії Google, що дозволяє використовувати голосовий пошук на основі технології розпізнавання мови. Ця технологія інтегрована в смартфони і комп'ютери з можливістю введення голосової інформації. Влітку 2011 року корпорація інтегрувала дану технологію в свою пошукову систему. На персональних комп'ютерах підтримується браузером Google Chrome. Також існує голосове управління для смартфонів з ОС Android. З травня 2014 року доступ до Google Speech Recognition API став легальний. Для роботи з базою даних системи розпізнавання необхідно зареєструватися в Google Developers. Голосовий пошук Google за замовчуванням впроваджений в багато популярних сервісів: Google, Yahoo, YouTube, Вікіпедія та ін.

Yandex Speech Kit – система розпізнавання мовлення від компанії Яндекс. Розробники запевняють, що даний набір засобів розробки є найкращим вибором для розпізнавання російської мови. Встановлено обмеження на кількість запитів (десять тисяч на добу). За словами топ-менеджерів компанії, таке рішення досить важко вивести на міжнародний ринок, так як з точки зору патентів багато із них належить Nuance. Ефективність розпізнавання в Yandex Speech Kit залежить, в першу чергу, від якості вхідного звуку, кодування, чіткості мовця, темпу розмови, складності

та довжини фраз. Розпізнавання мови відбувається в реальному часі разом з передачею звукової інформації, затримка не перевищує однієї секунди. Для забезпечення такої високої швидкості технологія працює в режимі потокового розпізнавання з проміжними результатами. Це означає, що як тільки людина починає говорити, її мова починає передаватися на сервіс невеликими частинами.

Вищеперечислені системи мають закритий вихідний код. Розглянемо систему з відкритим вихідним кодом CMU Sphinx. Саме вона буде використовуватись в подальшій роботі.

CMU Sphinx (або просто Sphinx) була створена групою розробників з університету Карнегі-Меллон. Система складається з набору бібліотек для розпізнавання мови (Sphinx 2–4) і акустичної моделі (Sphinx train).

Sphinx є розпізнавачем безперервної мови, що працює з використанням Прихованих Марківських моделей і статистичних мовних моделей. В системі реалізовані можливості розпізнавання довгих по часу промов і для 15 мов (українська туди не входить) створені великі словники для розпізнавання. З кожним виходом нової версії, Sphinx значно покращує свої результати в плані ефективності.

Sphinx 4 – найновіша версія бібліотеки Sphinx, що забезпечує базу для дослідження технологій розпізнавання мовлення. Написана на мові програмування Java. У майбутньому розробники планують створення і розвиток нових акустичних моделей, реалізацію системи мовної адаптації, удосконалення системи керування конфігураціями.

1.2.3 Основні проблеми розпізнавання мовлення

За останнє десятиліття в розвитку розпізнавання відбувся різкий ривок вгору. Але тим не менш, машина ще не досягла тієї точності розпізнавання, яку має звичайна людина. Існує декілька складнощів, над якими ще треба працювати та добиватись покращень.

Одна з ключових проблем розпізнавання мовлення – обробка акцентів, діалектів і фонового шуму. Причому збільшення тренувальних даних аж ніяк не вирішує цю проблему. Існують мови з безліччю акцентів та діалектів, для них практично нереально зібрати дані. Тільки для створення високоякісного розпізнавача однієї мови потрібно близько 5 тисяч годин аудіозаписів, з текстовим перекладом. Також, при наявності невеликого рівня шуму, люди не перестають чути та розуміти одне одного, в той час як машини різко втрачають якісні показники.

Ще однією проблемою є семантичні помилки, коли система неправильно розпізнає сенс фрази. Нещодавно працівники з Microsoft дослідили людські та машинні помилки, і зробили висновок, що більшість помилок однакового типу. За статистикою, у звичайній повсякденній розмові людина не розуміє 1 з 20 слів, тобто 5%. Це досить великий відсоток, але, на щастя, окрім звукового потоку ми використовуємо додаткові ознаки, які допомагають нам розпізнати слова. Наприклад, ми знаємо тему розмови, володіємо інформацією про людину, з якою ми спілкуємось, і при цьому використовуємо візуальні підказки (емоції на обличчі, рухи губ та тіла). В сукупності це все можна назвати як розпізнавання з урахуванням контексту. Нові системи розпізнавання поступово включають такі сигнали в свої алгоритми. Наприклад, голосовий асистент в ОС Android використовує список контактів для кращого розпізнавання імен, які там записані. Також, для того, аби легше побудувати маршрут, голосовий пошук на картах використовує геолокацію пристрою. Але таких прикладів поки що небагато, зараз лише починають впроваджувати подібні методи у системи розпізнавання.

Хороший розпізнавач повинен розділяти аудіопотік на однорідні сегменти у відповідності до спікера, аби позбутися проблеми багатоголосся в одному каналі. Окрім цього, він має перетворювати ці сегменти в текст, і на виході отримати декілька окремих блоків тексту.

Окрім стійкості розпізнавача до шуму та акцентів, він має бути стійким до ефекту ехо, форматів запису мовлення, обладнання і т.д. Людському вуху важко розрізнити різні формати записів, в той час як для розпізнавача це можуть бути кардинально різні дані.

Отже, можна зробити висновок, що задача розпізнавання мовлення на сьогоднішній момент є до кінця не розв'язаною. Залишається ще чимало невирішених і складних проблем. Можна відмітити такі:

- розпізнавання діалектів, акцентів мови
- розпізнавання на тлі сильного шуму
- включення контексту в процес розпізнавання
- поділ багатоголосного потоку
- зменшення кількості семантичних помилок
- зменшення затримки – часу від закінчення промови користувача та до закінчення отримання транскрипції

1.3 Формалізація постановки задачі

Перед нами стоїть задача розпізнавання української мови на основі Прихованих Марківських моделей. У якості практичного застосування буде представлено продукт, який дозволяє розпізнавати людські команди, що направлені комп'ютеру, та виконувати їх, або іншими словами, реалізація голосового управління комп'ютером (операційна система – Mac OS).

Поділимо основну задачу на підзадачі, отримуємо:

- підготовка словника з необхідними для розпізнавання українськими словами
- G2P (Grapheme-to-phoneme) – система конверсії слів у послідовності фонем
- запис вхідних даних з мікрофона
- виділення ознак з вхідного звукового потоку

- розпізнання звуків за допомогою акустичної моделі
- співставлення послідовності звуків словам зі словника
- виконання розпізнаних системних команд

1.4 Висновки

У даному розділі було розглянуто актуальність задачі розпізнавання української мови та приклади можливості використання у різних сферах життя. Очевидно, що задача має високий потенціал, сьогодні активно ведуться розробки у цій області багатьма компаніями. Було приділено увагу історії розвитку систем розпізнавання, та сучасним тендеціям. Було проаналізовано декілька сучасних платформ розпізнавання з закритим та відкритим вихідним кодом та обрано одну із них для подальшого використання у роботі, а саме систему CMU Sphinx. Також розглянуто декілька основних проблем, з якими сьогодні зіштовхуються системи розпізнавання та визначено напрям, у якому будуть рухатись ці розробки.

В кінці розділу були сформульовані основні задачі для подальшої роботи.

2 ПРОЦЕС РОЗПІЗНАВАННЯ МОВЛЕННЯ

2.1 Запис і оцифрування мови

Записані звукові дані подаються у вигляді аналогових сигналів. З даними такого типу система розпізнавання працювати не може. Тому перед безпосередньою обробкою звуку, його необхідно перевести у послідовність цифрових сигналів. Це етап попередньої обробки.

2.2 Виділення ознак

На цьому етапі у цифровому сигналі виділяються ділянки, які містять розмовну частину, а також знаходяться параметри ознак мовлення. Основна ціль даного етапу полягає в тому, щоб зберегти корисні дані та відсіяти непотрібну інформацію.

Спершу дані розбиваються на фрейми – невеликі часові проміжки. Причому фрейми повинні йти не один за одним, а частково перекриватись. Тобто кінцева частина одного фрейму має перетинатись з початковою частиною наступного фрейму. Робота з фреймами дозволяє аналізувати звукові дані більш ефективніше, ніж якщо б ми використовували значення цифрового сигналу в окремих точках. А перекриття фреймів один одним дозволяє згладити вихідні результати аналізу. На практиці встановлено, що оптимально розбивати звук на фрейми довжиною 25 мс з кроком 10 мс.

Для більш інформативного представлення, з фреймів виділяють ознаки. Існує декілька способів представлень таких ознак, найпопулярнішими є: Мел-частотні кепстральні коефіцієнти (Mel-Frequency Cepstral Coefficient – MFCC), Лінійні прогнозовні кепстральні коефіцієнти (Linear Predictive Cepstral Coefficient – LPCC), коефіцієнти лінійного пророкування Perceptual Linear Prediction (PLP), Filter-bank features, вейвлет-коефіцієнти.

2.2.1 MFCC ознаки

Такі коефіцієнти є своєрідним представленням енергії спектру сигналу. MFCC коефіцієнти мають свої переваги, а саме:

- використання безпосередньо спектру сигналу, а саме розкладання по базису ортогональних (ко)синусоїдальних функцій. Це допомагає врахувати хвильову "природу" сигналу при подальшому аналізі
- створення проекції спектру на спеціальну mel-шкалу для того, щоб виділити найбільш помітні частоти для людського сприйняття
- кількість коефіцієнтів у вихідному результаті може бути обмежена будь-яким значенням (зазвичай, 13). За рахунок цього фрейм можна стиснути і таким чином зменшити кількість оброблюваної інформації, коли це необхідно

Алгоритм знаходження MFCC-коефіцієнтів для заданого фрейму

Нехай наш фрейм – це вектор $x[k], 0 \leq k \leq N$, де N – розмір фрейму.

1) Розклад у ряд Фур'є

Потрібно розрахувати спектр сигналу за допомогою дискретного перетворення Фур'є. Для цього можна використати FFT-реалізацію:

$$X[k] = \sum_{n=0}^{N-1} x[n] * e^{\frac{-2\pi i k n}{N}}, \quad 0 \leq k \leq N \ (1)$$

Застосуємо віконну функцію Хеммінга для зглажування значень на межах фреймів:

$$H[k] = 0.54 - 0.46\cos(\frac{2\pi k}{N-1}) \ (2)$$

Результатом буде вектор наступного вигляду:

$$X[k] = X[k] * H[k], 0 \leq k < N \text{ (3)}$$

2) Обчислення mel-фільтрів

Mel – це "психофізична одиниця висоти звуку", заснована на суб'єктивному сприйнятті людиною. В першу чергу вона залежить від частоти звуку (а також від гучності і тембру). Грубо кажучи, ця величина показує, на скільки звук певної частоти "значущий" для людського вуха.

Перетворення частоти в mel-одиницю виконується наступною формулою:

$$M = 1127 * \log(1 + \frac{F}{700}) \quad (4)$$

Обернене перетворення має наступний вигляд:

$$F = 700 * (e^{\frac{M}{1127}} - 1) \quad (5)$$

Графік залежності mel-одиниць від частот зображено на рис. 1:

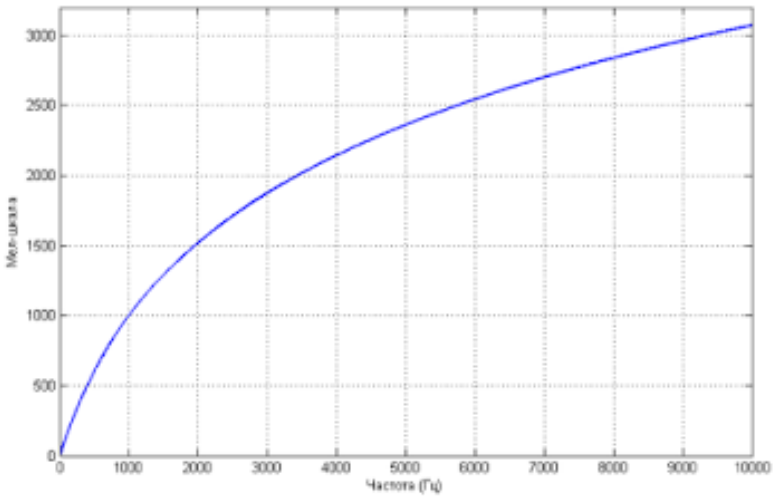


Рисунок 1 – Графік залежності mel-одиниць від частот

Для розкладання спектру сигналу по mel-шкалі необхідно побудувати mel-фільтри. Кожен mel-фільтр це трикутна віконна функція, яка дозволяє порахувати кількість енергії на певному діапазоні частот і таким чином отримати mel-коефіцієнт. Знаючи кількість mel-коефіцієнтів і діапазон частот можна побудувати набір наступних фільтрів (рис. 2):

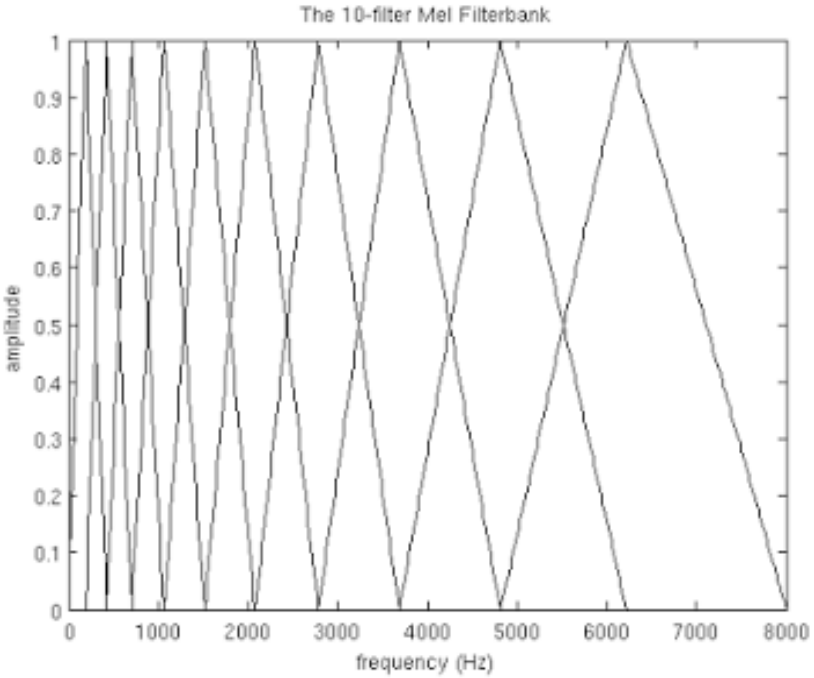


Рисунок 2 – Графік mel-фільтрів

З графіку видно – чим більше порядковий номер mel-коефіцієнта, тим ширше основа фільтра. Це пов'язано з тим, що розбиття необхідного діапазону частот на фільтри відбувається на mel-шкалі.

Для кращого представлення розглянемо приклад.

Нехай ми маємо фрейм з 256 елементами з частотою звуку 16000 Hz. Припустимо, що людська мова лежить в діапазоні від 300 до 8000 Hz. Кількість mel-коефіцієнтів, які ми хочемо знайти, рівна 10.

Застосувавши формулу (4) діапазон [300;8000] Hz зменшився до [401.25; 2834.99] Hz.

Для побудови 10 трикутних фільтрів необхідно взяти 12 опорних точок:

m[i] = [401.25, 622.50, 843.75, 1065.00, 1286.25, 1507.50, 1728.74, 1949.99, 2171.24, 2392.49, 2613.74, 2834.99]

Важливо, що точки на mel-шкалі мають рівномірний розподіл.

Застосуємо формулу (5) для оберненого перетворення шкали в Hz.

h[i] = [300, 517.33, 781.90, 1103.97, 1496.04, 1973.32, 2554.33, 3261.62, 4122.63, 5170.76, 6446.70, 8000]

Бачимо, що шкала поступово розтягується, вирівнюючи тим самим динаміку зростання "значущості" на низьких і високих частотах.

Накладемо отриману шкалу на спектр нашого фрейму. По осі X у нас знаходиться частота. Довжина спектра 256 елементів, при цьому в ньому вміщується 16000 Hz. Обрахувавши пропорцію отримуємо наступну формулу:

$$f(i) = floor((frameSize + 1) * \frac{h(i)}{sampleRate}) \quad (6)$$

В нашому прикладі отримуємо такий результат:

f(i) = 4, 8, 12, 17, 23, 31, 40, 52, 66, 82, 103, 128

Знаючи опорні точки на осі X спектру, легко побудувати необхідні фільтри за такою формулою:

$$H_m(k) = \begin{cases} 0, & k < f(m-1) \\ \frac{k - f(m-1)}{f(m) - f(m-1)}, & f(m-1) \le k \le f(m) \\ \frac{f(m+1) - k}{f(m+1) - f(m)}, & f(m) \le k \le f(m+1) \\ 0, & k > f(m+1) \end{cases} \quad (7)$$

3) Застосування фільтрів та логарифмування енергії спектра

Застосування фільтру полягає в попарному перемножуванні його значень зі значеннями спектра. Результатом цієї операції буде той же mel-коефіцієнт. Оскільки фільтрів у нас M, коефіцієнтів буде стільки ж.

$$S[m] = \log(\sum_{k=0}^{N-1} |X[k]|^2 * H_m[k]), 0 \leq m \leq M \quad (8)$$

Однак, mel-фільтри потрібно застосувати до енергії спектра, а не до його значень. Після чого прологарифмувати отримані результати. Таким чином ми знижуємо чутливість коефіцієнтів до шумів.

4) Косинусне перетворення

За допомогою дискретного косинусного перетворення (DCT) можна отримати "кепстральні" коефіцієнти. Основна його задача – це "стиснення" отриманих результатів, причому з підвищенням значущості перших коефіцієнтів і зменшення значущості останніх.

DCT рахуємо за такою формулою:

$$C[l] = \sum_{m=0}^{M-1} S[m] * \cos(\pi * l * \frac{m+\frac{1}{2}}{M})), \ 0 \leq l < M \quad (9)$$

У результаті для кожного фрейму ми отримаємо набір MFCC-коефіцієнтів розміру M. Надалі їх можна буде використати у якості вхідних даних для акустичної моделі.

2.3 Акустичне моделювання

Акустичне моделювання є основною частиною системи розпізнавання мовлення. Саме за допомогою нього дається відповідь на питання, яка фонема була озвучена на заданому фреймі. Найчастіше, відповідь не може бути надана однозначно, тому доводиться відповідати у термінах ймовірностей – тобто, наприклад, для даного сигналу одні фонеми можна розглядати з більшою ймовірністю, а деякі одразу відкидати. Строго кажучи, акустична модель - це функція, яка приймає на вхід деяку ділянку звукового сигналу (фрейм) та, як результат, повертає розподіл ймовірностей фонем на цьому фреймі.

Сучасні методи та алгоритми розпізнавання мовлення можна поділити на наступні класи:

- Динамічне програмування - алгоритм динамічної трансформації часової шкали (Dynamic Time Wrapping - DTW).
- Приховані Марківські моделі (Hidden Markov Models - HMM).
- Нейронні мережі (Neural Networks - NN).

Розглянемо кожен з вищеперечислених методів детальніше.

2.3.1 DWT алгоритм

У даному алгоритмі для визначення слів порівнюються числові представлення цифрових сигналів мовлення або їх спектограми. При цьому мають бути компенсовані різні довжини послідовності і нелінійний характер звуку. Ці проблеми вирішуються шляхом знаходження деформації, яка відповідає оптимальній відстані між двома рядами різної довжини.

Існує два підходи застосування алгоритму:

1. Пряме порівняння числових форм сигналів. В цьому випадку, для кожної числової послідовності створюється нова послідовність, розміри якої значно менше. Числова послідовність може мати кілька тисяч числових значень, в той час як підпослідовність може мати кілька сотень значень.

Зменшення кількості числових значень може бути виконано шляхом їх видалення між кутовими точками. Цей процес скорочення довжини числової послідовності не повинен змінювати свого уявлення. Безсумнівно, процес призводить до зменшення точності розпізнавання. Однак, беручи до уваги збільшення швидкості, точність, по суті, підвищується за рахунок збільшення слів в словнику.

2. Подання сигналів спектрограм і застосування алгоритму DTW для порівняння двох спектрограм. Метод полягає в поділі цифрового сигналу на деяку кількість інтервалів, які будуть перекриватися. Для кожного імпульсу, інтервали дійсних чисел (звукових частот), будуть розраховуватись швидким

перетворенням Фур'є, і зберігатися у матриці звукової спектрограми. Параметри будуть однаковими для всіх обчислювальних операцій: довжин імпульсу, довжини перетворення Фур'є, довжини перекриття для двох послідовних імпульсів. Перетворення Фур'є є симетрично пов'язаним з центром, а комплексні числа з одного боку пов'язані з числами з іншого боку.

У зв'язку з цим, тільки значення з першої частини симетрії можна зберегти, таким чином, спектрограма представлятиме матрицю комплексних чисел, кількість ліній в такій матриці є рівною половині довжини перетворення Фур'є, а кількість стовпців буде визначатися в залежності від довжини звуку. DTW буде застосовуватися на матриці дійсних чисел в результаті сполучення спектрограми значень, така матриця називається матрицею енергії.

DTW застосовується до відео, аудіо та графіки – дійсно, будь-які дані, які можна подати лінійним представленням, можна проаналізувати за допомогою DTW.

Цей підхід історично використовувався для розпізнавання мови, але в даний час значною мірою витісняється більш успішним підходом на основі ПММ.

2.3.2 Приховані Марківські моделі

Для визначення прихованої Марківської моделі необхідно ввести наступні елементи:

1) N – число станів моделі. Для позначення множини станів моделі використовується запис $S = \{S_1, S_2,\ldots,S_N\}$, а стан моделі в момент t позначається q_t.
2) M – число символів спостережень, які можуть бути породжувані моделлю, або іншими словами, розмір дискретного алфавіту. Символи спостережень відповідають фізичному виходу

модельованої системи. Множина спостережуваних символів позначається як $V = \{v_1, v_2, \ldots, v_M\}$.

3) Розподіл ймовірностей переходів між станами (або матриця ймовірнісних переходів) $A = \{a_{ij}\}$, де:

$$a_{ij} = P[q_{t+1} = S_j | q_t = S_i],\ 1 \le i,j \le N. \quad (10)$$

У випадку, коли кожен стан може бути досягнутий з будь-якого іншого за один крок, матриця $a_{ij}>0$ для всіх i, j.

4) Розподіл ймовірностей появи символів спостереження у стані j, $B = \{bj(k)\}$, де:

$$b_j(k) = P[v_k \text{ at } t | q_t=S_j],\ 1 \le j \le N,\ 1 \le k \le M. \quad (11)$$

5) Початковий розподіл ймовірностей станів $\pi = \{\pi_i\}$:

$$\pi_i = P[q_i = S_i],\ 1 \le i \le N. \quad (12)$$

Обравши значення для N, M, A, B, π, можна використовувати ПММ для генерування деякої послідовності спостережень $O = O_1O_2...O_T$, де кожне спостереження O_t є символом алфавіту V, а T – число символів у спостережуваній послідовності), яку можна отримати наступною процедурою:

- обрати початковий стан $q_1 = S_i$ у відповідності з розподілом π;
- встановити $t = 1$;
- обрати $O_t = v_k$ у відповідності з розподілом ймовірностей появи символів алфавіту у стані S_i, тобто у відповідності з $b_i(k)$;
- перейти до нового стану $q_{t+1}=S_j$ у відповідності з перехідними ймовірностями a_{ij};

– встановити t = t+1, і якщо t < T, повернутись до кроку 3, інакше закінчити процедуру.

З вищесказаного слідує, що повний опис ПММ передбачає задання двух параметрів моделі N, M, множини допустимих символів спостереження, а також трьох ймовірнісних пар A, B, π. Надалі для позначення всієї множини параметрів буде використовуватись запис $\lambda = (A, B, \pi)$.

Відповідно до опису прихованої марковської моделі, поданому вище, можна поставити три питання, на які має бути надана відповідь, аби модель могла успішно вирішувати поставлені перед нею завдання.

1) Нехай задана послідовність спостережень $O = O_1O_2...O_T$ та модель $\lambda = (A, B, \pi)$. Як ефективно обчислити величину $P(O|\lambda)$ – ймовірність появи цієї послідовності спостережень для заданої моделі?

2) Як обрати оптимальну послідовність станів $Q = q_1, q_2, \ldots, q_T$, (тобто ту, яка найкращим чином відповідатиме відповідній існуючій послідовності спостережень)?

3) Яким чином необхідно побудувати параметри моделі $\lambda = (A, B, \pi)$ для того, аби максимізувати $P(O|\lambda)$?

Питання 1 є звичайною задачею оцінювання моделі. Відомі моделі та послідовності спостережень, необхідно обчислити ймовірність того, що спостереження породжені саме заданою моделлю. Можна розглянути це питання також як задачу вияснення наскільки добре обрана модель відповідає заданій послідовності спостережень. Такий підхід має велику практичну цінність. Наприклад, якщо у нас стоїть питання вибору найкращої моделі з набору вже існуючих, то рішення першої задачі дозволяє отримати відповідь на це запитання.

Відповідь на друге запитання полягає в тому, щоб зрозуміти що відбувається в прихованій частині моделі, тобто знайти «правильну» послідовність станів, яку проходить модель та відкрити “приховану” частину моделі. Очевидно, що при заданій послідовності спостережень та відомій моделі неможливо знайти абсолютно точну послідовність станів, які

відповідають спостереженням, за винятком того виродженого випадку, коли задана послідовність спостережень породжена саме цієї моделлю. Тут можна говорити лише про припущення з відповідною ступенем достовірності. Тому на практиці для наближеного вирішення цієї проблеми ми будемо використовувати деякі критерії оптимальності. Далі ми побачимо, що існує одразу декілька критеріїв оптимальності для визначення послідовності станів. Тому вибір одного з них буде в значній мірі залежати від цілей подальшого використання отриманої послідовності спостережень. В нашому випадку ціль – це пошук оптимальної послідовності станів для розпізнавання спонтанного мовлення і т.д.

Третє питання представляє собою задачу, в якій ми намагаємось оптимізувати значення параметрів, щоб модель найкращим чином відповідала попереднім спостереженням. Послідовність спостережень, яка використовується для налаштування параметрів називається навчальною послідовністю, оскільки вона використовується саме для навчання ПММ. Проблема навчання дуже важлива для більшості застосунків, так як саме під час навчання відбувається, у відповідності з обраним критерієм оптимальності, налаштування значень параметрів моделі за даними спостережень. В результаті цього будується модель, яка найкращим чином описує реальні процеси.

Розв'язок першого питання

Отже, за заданою моделлю λ необхідно знайти ймовірність появи послідовності спостережень $O = O_1O_2...O_T$, тобто $P(O|\lambda)$. Найбільш очевидний алгоритм – перечислити всі можливі послідовності станів довжини T (тобто числа спостережень в послідовності). Зафіксуємо одну таку послідовність станів $Q = q_1, q_2, \ldots, q_T$, де q_1 – початковий стан. Ймовірність появи послідовності спостережень O для послідовності вищезгаданих станів при умові незалежності спостережень визначається формулою:

$$P(O|Q, \lambda) = \prod_{t=1}^{T} P(O_t|q_t, \lambda) \text{ . (13a)}$$

Звідси отримуємо:

$$P(O|Q, \lambda) = b_{q1}(O_1)*b_{q2}(O_2)*...*b_{qT}(O_T). \quad (13б)$$

Ймовірність такої послідовності станів Q можна записати у вигляді:

$$P(O|\lambda) = \pi_{q1}a_{q1q2}a_{q2q3}...a_{qT\text{-}1qT}. \quad (14)$$

Сумісна ймовірність послідовностей O та Q, тобто ймовірність появи O та Q разом – це просто добуток ймовірностей (13а) та (13б):

$$P(O, Q|\lambda) = P(O|Q, \lambda)P(Q, \lambda). \quad (15)$$

Ймовірність появи послідовності спостережень O (для даної моделі) отримується шляхом сумування цих сумісних ймовірностей для всіх можливих послідовностей Q, що дає:

$$P(O|\lambda)=\sum_{all\ Q} P(O|Q,\lambda)*P(Q|\lambda)=\sum_{q1q2\ldots qT} \pi_{q1}b_{q1}(O_1)a_{q1q2}b_{q2}(O_2)...a_{qt\text{-}1qt}b_{qt}(Q_T). \quad (17)$$

Обчислення, подані вище, можна інтерпретувати наступним чином. Спочатку в момент часу t = 1 модель знаходиться у стані q_1 з ймовірністю π_{q1} і породжує в цьому стані з ймовірністю $b_{q1}(O_1)$ символ O_1. Час змінюється з t на t + 1 (тобто стає рівним 2), і модель з ймовірністю a_{q1q2} виконує перехід із стану q_1 у стан q_2, генеруючи при цьому символ O_2 з ймовірністю $b_{q2}(O_2)$. Процес продовжується до тих пір, доки не буде виконаний останній перехід у момент часу T із стану $q_{T\text{-}1}$ з ймовірністю $a_{qT\text{-}1qT}$ у стан q_T, у якому з ймовірністю $b_{qT}(O_T)$ буде згенерований символ O_T.

Неважко порахувати, що обчислення ймовірності P(O|λ) у відповідності з визначенням (17) потребує $2T*N^T$ операцій, так як у кожен момент часу t = 1, 2, …, T існує N^T різних можливих станів, а отже, існує N різних

послідовностей станів, тому для обчислення одного члену суми (17) необхідно приблизно 2*T операцій (точніше кажучи, потрібно (2T - 1)N множень та N - 1 сум). Навіть при малих значенннях N та T такий розрахунок провести досить складно. Наприклад, при N = 5 і T = 100 потрібно приблизно 2 * 100 * 5100 ≈ 1072 операцій. Для цього використовується більш ефективна процедура – алгоритм прямого - оберненого ходу (the forward - backward procedure).

Алгоритм прямого - оберненого ходу

Введемо змінну $\alpha_t(i)$, визначену наступним чином:

$$\alpha_t(i) = P(O_1O_2...O_t, q_t = S_i|\lambda), \quad (18)$$

яка представляє собою ймовірність появи для даної моделі часткової послідовності спостережень $O_1O_2...O_t$ (до моменту t і стану в цей момент). За індукцією можна обчислити $\alpha_t(i)$:

1) ініціалізація

$$\alpha_t(i) = \pi_i b_i(O_1), \ 1 \le i \le N. \quad (19)$$

2) індуктивний перехід

$$\alpha_{t+1}(j) = [\sum_{i=1}^{N} \alpha_t(j)a_{ij}]b_j(O_{t+1}), \ 1 \le t \le T - 1, \ 1 \le j \le N. \quad (20)$$

3) закінчення

$$P(O|\lambda) = i = \sum_{i=1}^{N} \alpha_T(i) \quad (21)$$

Крок 1 встановлює значення змінної, яке дорівнює сумісній ймовірності стану і початкового спостереження O_1. Індуктивний перехід, в

якому заключається основний зміст прямого ходу обчислень, показаний на рис. 3:

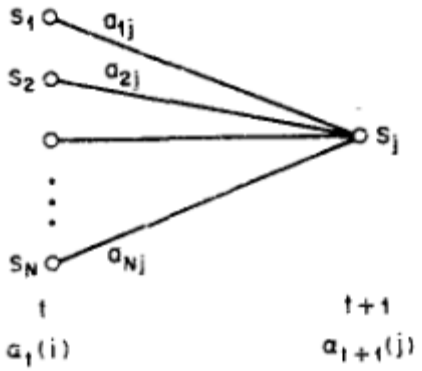


Рисунок 3 – перехід від N можливих станів S_i в S_j в момент t

На ньому видно, як саме стан Sj може бути досягнутий в момент t+1 з N можливих станів S_i, $1 \leq i \leq N$ у момент t. Так як $\alpha_t(i)$ це ймовірність події, яка полягає в тому, що спостерігається послідовність $O_1O_2...O_t$, причому стан в момент t є S_i, то добуток $\alpha_t(i)a_{ij}$ представляє собою ймовірність того, що для спостережень $O_1O_2...O_t$ і стану S_i в момент t, в наступний момент t + 1 буде досягнутий стан S_j. Сума цього добутку по всім можливим станам S_i, $1 \leq i \leq N$, в момент часу t дає ймовірність появи стану S_j в момент t + 1 сумісно зі всіма попередніми спостереженнями. Визначивши S_j , неважко переконатись в тому, що $\alpha_{t+1}(j)$ отримується шляхом обчислення ймовірності спостереження O_{t+1} у стані j, тобто множенням результату сумування на ймовірність $b_j(O_{t+1})$. Обчислення у відповідності з (20) при фіксованому t виконуються для всіх станів j, $1 \leq j \leq N$, і повторюються при t = 1,2, …, T-1. Нарешті крок 3 приводить до обчислення шуканої ймовірності $P(O|\lambda)$ як суми кінцевих значень прямих змінних $\alpha_T(i)$. Це випливає з того, що по означенню:

$$\alpha_T(i) = P(O_1O_2...O_T, q_T = S_i|\lambda) \quad (22)$$

і відповідно $P(O|\lambda)$ є сумою по і величинам $\alpha_T(i)$.

Для обчислення всіх $\alpha_T(i)$, $1 \le i \le N$, необхідно N^2T операцій, що значно менше за число $2T*N^T$ операцій, які необхідні при прямих обчисленнях за формулами (16)-(17).

Аналогічним чином можна ввести обернену змінну $\beta_t(i)$, яка визначається за формулою:

$$\beta_t(i) = P(O_{t+1}O_{t+2}...O_T|q_t=S_i, \lambda), \quad (23)$$

яка для заданої моделі λ представляє собою сумісну ймовірність появи часткової послідовності спостережень від моменту часу t+1 до T і стану S_i в момент часу t. Як і раніше, можна обчислити $\beta_t(i)$ за допомогою індукції:

1) ініціалізація

$$\beta_T(i) = 1, \ 1 \le i \le N. \quad (24)$$

2) індуктивний перехід

$$\beta_{t+1}(i) = j = \sum_{i=1}^{N} \alpha_{ij} b_j(O_{t+1}), \ T - 1 \ge t \ge 1, \ 1 \le i \le N. \quad (20)$$

Крок 1 довільно визначає $\beta_T(i) = 1$ для всіх i.

Крок 2 проілюстровано на рис. 4:

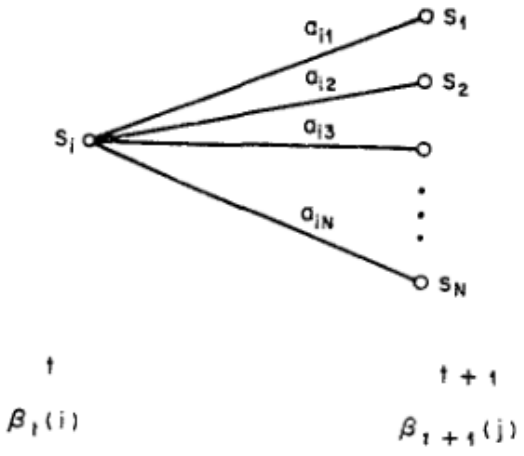


Рисунок 4 – Перехід від стану S_i у момент t до можливих станів S_j, $1 \le j \le N$

Видно, що для обчислення ймовірності того, що при заданій послідовності спостережень в моменти часу від t+1 до T модель в момент t знаходилась у стані S_i, необхідно розглянути всі можливі стани S_j в момент t+1, врахувати ймовірності переходів з S_i в S_j (або a_{ij}), ймовірність спостереження O_{t+1} у стані j (або $b_j(O_{t+1})$), а також ймовірність послідовності спостережень зі стану j(або $\beta_{t+1}(j)$). Ці обчислення ми зможемо ефективно використати для вирішення наступних питань.

На відміну від питання 1, для якого можна дати точне рішення, для питання 2, а саме пошуку оптимальної послідовності станів, яка відповідає заданій послідовності спостережень, існує декілька можливих рішень. Складність визначення оптимальної послідовності станів обумовлена тим, що існує декілька можливих критеріїв оптимальності. Наприклад, суть одного з можливих критеріїв полягає в тому, щоб обрати такі стани q_t, кожен з яких, взятий окремо, є найбільш ймовірним. Цей критерій оптимальності максимізує число коректно визначених індивідуальних станів. Для того, щоб розв'язати проблему 2 на основі цього критерію, необхідно ввести змінну:

$$\gamma_t(i) = P(q_t = S_i|O, \lambda), \quad (26)$$

яка представляє собою ймовірність перебування в момент t в стані S_i при заданій послідовності спостережень O і моделі λ. Використовуючи пряму та обернену змінні, рівняння (26) можна представити в наступному вигляді:

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(O|\lambda)} = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N}\alpha_t(i)\beta_t(i)} \quad (27)$$

де $\alpha_t(i)$ відповідає частковій послідовності спостережень $O_1O_2...O_t$ і стану S_i в момент t, а $\beta_t(i)$ – залишку послідовності спостережень $O_{t+1}O_{t+2}...O_T$ і заданому стану S_i в момент t. Нормуючий множник $P(O, \lambda) = \sum_{i=1}^{N}\alpha_t(i)\beta_t(i)$ обраний таким чином, щоб ймовірнісна міра $\gamma_t(i)$ задовольняла умові:

$$\sum_{i=1}^{N} \gamma_t(\mathrm{i}) = 1. \quad (28)$$

Використовуючи $\gamma_t(i)$, можна обчислити найбільш ймовірний стан q_t в момент t як стан, визначений виразом:

$$q_t = \mathrm{argmax}_{1 \le i \le N} \ [\gamma_t(i)], \ 1 \le t \le T. \quad (29)$$

І хоча рівняння (29) максимізує очікуване число правдоподібних станів (оскільки в кожен момент часу t обирається найбільш ймовірний стан), з отриманої таким чином послідовності станів може виникнути ряд труднощів. Так, наприклад, якщо деякі перехідні ймовірності ПММ дорівнюють 0 (тобто $a_{ij} = 0$ для деяких i, j), знайдена оптимальна ймовірність станів може виявитись недопустимою послідовністю. Обумовлено це тим, що рівняння (29) просто визначає найбільш ймовірний стан для кожного моменту часу і не враховує ймовірність появи послідовностей станів.

Один з можливих варіантів розв'язати питання полягає в модифікації критерію оптимальності. Наприклад, можна розв'язати задачу пошуку послідовності станів, яка максимізує число коректних пар станів (q_t, q_{t+1}) або трійок станів (q_t, q_{t+1}, q_{t+2}). Хоча в деяких випадках використання цих критеріїв і має сенс, але все ж найбільш широко використовується критерій, який базується на пошуку єдиної найкращої послідовності станів, яка максимізує P(O, λ), що еквівалентно P(Q, O|λ). Метод визначення такої найкращої послідовності, заснований на алгоритмі динамічного програмування, який має назву алгоритм Вітербі.

Алгоритм Вітербі

Для того, щоб по заданій послідовності спостережень $O = (O_1O_2...O_T)$ знайти найкращу послідовність станів $Q=\{q_1q_2...q_T\}$ визначимо наступну величину:

$$\delta_t(i) = \max_{q1,q2\ldots qt-1} P[q_1 q_2 ... q_t] = i, O_1 O_2 ... O_t|\lambda), \quad (30)$$

яка представляє собою максимальну ймовірність того, що при деяких заданих t перших спостереженнях послідовність станів у відповідні моменти часу закінчується (в момент t) у стані S_i. За індукцією отримуємо:

$$\delta_{t+1}(i) = [\max_j \delta_t(i) a_{ij}] * b_j(O_{t+1}), \quad (31)$$

Для того, щоб надалі відновити послідовність станів для всіх значень t та j, необхідно зберігати значення аргументів, які максимізують ймовірність (31). Для цієї цілі ми будемо використовувати масив $\psi_t(j)$. Повну процедуру, потрібну для визначення послідовності станів, можна тепер сформулювати наступним чином:

1) ініціалізація:

$$\delta_1(i) = \pi_i b_i(O_1), \ 1 \le i \le N, \quad (32a)$$

$$\psi_1(i) = 0. \quad (32b)$$

2) рекурсія:

$$\delta_t(i) = \max_{1 \le i \le N} [\delta_{t-1}(i) a_{ij}] * b_j(O_t), \ 2 \le t \le T, \ 1 \le j \le N \quad (33a)$$

$$\psi_t(i) = \operatorname{argmax}_{1 \le i \le N} [\delta_{t-1}(i) a_{ij}], \ 2 \le t \le T, \ 1 \le j \le N \quad (33b)$$

3) закінчення:

$$P^* = \max_{1 \le i \le N} [\delta_T(i)], \quad (34a)$$

$$q_T^* = \operatorname{argmax}_{1 \le i \le N} [\delta_T(i)]. \quad (34b)$$

4) відновлення шляху (послідовності станів):

$$q_t^* = \psi_{t+1}(q^*_{t+1}),\ t = T - 1, T - 2, \ldots, 1. \quad (35)$$

Необхідно відмітити, що реалізація алгоритму Вітербі аналогічна (за винятком кроку відновлення) прямому ходу обчислень (19) – (21). Основа різниця – використання замість процедури сумування (20) процедури максимізації за всіма попередніми станами (33а).

Третя, найбільш складна проблема ПММ – за заданою послідовністю спостережень визначити метод такого налаштування параметрів (A, B, π) моделі, щоб для отриманої модифікованої моделі ймовірність появи цієї послідовності була максимальною. Не існує відомого аналітичного виразу для параметрів цієї моделі. Крім того, на практиці, маючи деяку послідовність спостережень у якості навчальних даних, не можна вказати оптимальний спосіб оцінки параметрів. Тим не менш, використовуючи ітеративні процедури, наприклад метод Баума-Уелча, ЕМ-метод (Expectation-maximization) чи градієнтні методи, можна обрати параметри моделі $\lambda = (A, B, \pi)$ таким чином, щоб локально максимізувати ймовірність $P(O, \lambda)$. У цьому підрозділі розглядається ітеративна процедура вибору параметрів, побудована на класичні роботі Баума та його колег.

Для того, щоб описати процедуру переоцінки (тобто послідовного оновлення даних і відповідного налаштування параметрів) параметрів ПММ, визначимо величину $\xi_t(i,j)$ – ймовірність того, що при заданій послідовності спостережень в моменти часу t та t + 1 система буде відповідно знаходитись у станах S_i та S_j:

$$\xi_t(i,j) = P(q_t = S_i, q_{t+1} = S_j | O, \lambda). \quad (36)$$

Послідовність подій, які задовольняють (36) розглянуто на рис.5:

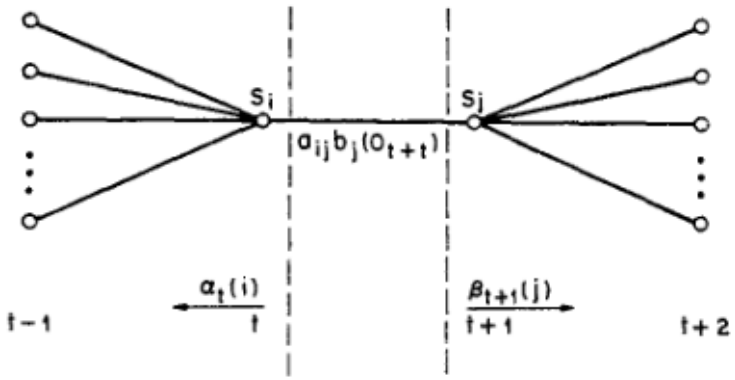


Рисунок 5 – Послідовність подій, де в моменти часу t та t + 1 система знаходиться у станах S_i та S_j

Використовуючи визначення прямої та оберненої змінних, вираз для $\xi_t(i,j)$ можна представити наступним чином:

$$\xi_t(i,j) = \frac{\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{P(O|\lambda)} = \frac{\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)} \quad (37)$$

де чисельник – це ймовірність $P(q_t = S_i, q_{t+1} = S_j, O|\lambda)$, а ділення на $P(O|\lambda)$ можна розглядати як нормування, для забезпечення необхідної вірогідної міри.

Вище $\gamma_t(i)$ було зазначено як ймовірність перебування моделі в момент t у стані S_i при заданій послідовності спостережень. Відповідно, $\gamma_t(i)$ можна записати через $\xi_t(i,j)$, сумуючи цю величину по j:

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j). \quad (38)$$

Якщо величину $\gamma_t(i)$ просумувати по всім t, то результат можна розглядати як очікуваний час перебування системи у стані S_i, або як очікуване число переходів, зроблених із стану S_i (якщо виключити t = T з границь сумування). Аналогічно, результат сумування $\xi_t(i,j)$ по t (від t = 1 до t = T - 1) можна розглядати як очікуване число переходів зі стану S_i в S_j.

$\sum_{t=1}^{T-1} \gamma_t(i)$ – очікуване число переходів з S_i, (39а)

$\sum_{t=1}^{T-1} \xi_t(i,j)$ – очікуване число переходів з S_i в S_j. (39б)

Використовуючи приведені вище формули (і принцип підрахунку появи подій), можна побудувати метод переоцінки параметрів ПММ. Відповідні формули для розрахунку оцінок параметрів (A, B, π) будуть мати наступний вигляд:

$\bar{\pi}_t$ = очікувана частота (кількість разів) перебування в стані S_i у момент (t = 1) = $\gamma_t(i)$ (40а)

$$\bar{a}_{ij} = \frac{\text{очікуване число переходів зі стану } S_i \text{ у стан } S_j}{\text{очікуване число переходів із стану } S_i} = \frac{\sum_{t=1}^{T-1} \xi t(i,j)}{\sum_{t=1}^{T-1} \gamma t(i)}, \text{ (40б)}$$

$$\bar{b}_j(k) = \frac{\text{очікуване число переходів у стан j і спостереження символа } v_k}{\text{очікуване число переходів у стан j}} = \frac{\sum_{t=1}^{T-1} \gamma t(j)}{\sum_{t=1}^{T-1} \gamma t(j)}, \text{ (40в)}$$

Нехай λ = (A, B, π) – вихідна модель, а $\bar{\lambda}$ = ($\bar{A}$, $\bar{B}$, $\bar{\pi}$) – модель після переоцінки параметрів, причому $\bar{A}$, $\bar{B}$, $\bar{\pi}$ відповідають лівим частинам рівностей (40а) – (40в), а для розрахунку правих частин цих рівностей використовуються A, B, π. Баумом та його колегами було показано, що в цьому випадку або вихідна модель визначає точку екстремуму функції правдоподібності, і тоді λ = $\bar{\lambda}$, або модель $\bar{\lambda}$ більш правдоподібна, аніж модель λ, в тому сенсі, що $P(O|\bar{\lambda}) > P(O|\lambda)$, що означає, що нова модель знайдена.

Якщо ітеративно повторювати цю процедуру, використовуючи на кожному новому кроці значення параметрів моделі, отримані на попередньому кроці, то поступово ймовірність появи послідовності спостережень O буде зростати. Процедура буде продовжуватись до тих пір, доки не буде досягнути деяка гранична точка. Отримана в результаті оцінки параметрів моделів буде називатись оцінкою максимальної правдоподібності для ПММ. Варто відмітити, що алгоритм прямого-оберненого ходу дозволяє

відшукувати лише локальні максимуми і що в більшості задач, які представляють собою практичний інтерес, поверхня, на якій виконується оптимізація, має складну форму з великою кількістю локальних максимумів.

Формули для повторного оцінювання (40а)–(40в) можна отримати безпосередньо шляхом максимізації по $\bar{\lambda}$ (з використанням звичайних методів оптимізації з обмеженнями) або іншими словами – допоміжну функцію Баума:

$$Q(\lambda, \bar{\lambda}) = \sum_Q P(Q|O, \lambda) \log [P(O, Q|\bar{\lambda})]. \quad (41)$$

Максимізація функції $Q(\lambda, \bar{\lambda})$ призведе до зростання функції правдоподібності, тобто:

$$\max_{\bar{\lambda}} [Q(\lambda, \bar{\lambda})] = P(O|\bar{\lambda}) \geq P(O|\lambda). \quad (42)$$

Рано чи пізно функція правдоподібності досягає деякої точки.

Формули повторного оцінювання можна інтерпретувати як реалізація статичного ЕМ-алгоритму, в якому крок Е(обчислення мат. сподівання) – це обчислення допоміжної функції $Q(\lambda, \bar{\lambda})$, а крок М (модифікація) – це максимізація цієї функції по $\bar{\lambda}$.

2.3.3 Рекурентні нейронні мережі

При нашому звичному людському спілкуванні, ми запам’ятовуємо не лише сказане в даний момент слово, але і ті, що були сказані попередньо. Людський мозок не починає думати з порожнього листа кожен момент часу, ми будуємо контекстно-словесні залежності, самі не розуміючи цього.

Звичайні нейронні мережі та ПММ не мають такої властивості, що є значним недоліком для розпізнавання довгих фраз та речень. Для розв’язання

таких динамічних задач у 1997 році були створені рекурентні нейронні мережі. Це клас штучних нейронних мереж, з’єднання між вузлами якого утворюють орієнтований у часі граф та допомагають зберігати попередню інформацію.

Позначимо x = (x1,…, xT) як вектор вхідних даних, y = (y1, …, yT) – вихідний вектор та h = (h1, …, hT) – вектор прихованого шару. Стандартна рекурента нейронна мережа обчислює h та y, ітеративно обчислюючи наступні вирази, починаючи з t = 1 і до T:

$$h_t = H(W_{xh}x_t + W_{hh}h_{t-1} + b_h) \quad (43)$$

$$y_t = W_{hy}h_t + b_y \quad (44)$$

Тут W це матриця вагів (W_{xh} - матриця вхідних-прихованих вагів), b (від англ. bias) – вектор зміщення (b_h прихований вектор зміщення), а H – функція прихованого шару. H зазвичай представляється як сигмоїда.

У ситуації, коли відстань між актуальною інформацією та місцем, де вона знадобилась є невеликою, РНМ можуть навчитись використанню інформації з минулих ітерацій. Але трапляються випадки, коли необхідно більше контексту. На жаль, по мірі зростання відстані між необхідними даними якість роботи РНМ погіршується. В таких випадках, застосовується особливий різновид архітектури РНМ, що має назву Довга короткочасна пам'ять (ДКЧП), яка має здатність навчатись довготривалим залежностям. Основним компонентом ДКЧП є клітинка та її стан. Рисунок 6 детально ілюструє її вигляд:

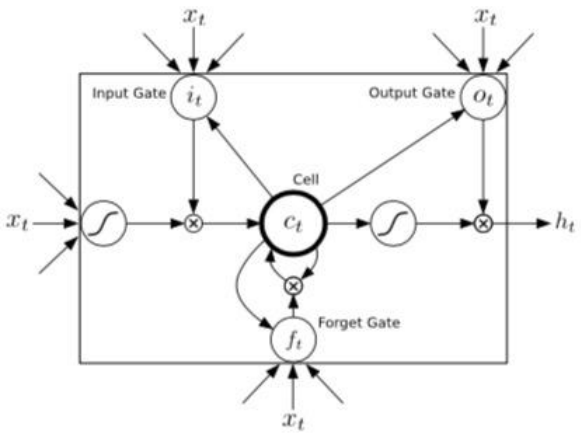


Рисунок 6 – Клітинка ДКЧП

ДКЧП задається такими обчисленнями:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \quad (45)$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \quad (46)$$

$$c_t = f_t c_{t-1} + i_t \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \quad (47)$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \quad (48)$$

$$h_t = o_t \tanh(c_t) \quad (49)$$

σ – це сигмоїдна функція

i – вектор вхідного вентиля (вага отримання нової інформації)

f – вектор забувального вентиля (вага пам’ятання старої інформації)

o – вектор вихідного вентиля (кандидатність на вхід)

c – вектор стану комірки

Вектори i, f, o,c мають таку ж розмірність як і прихований вектор h.

Однією з особливостей послідовних РНМ є те, що вони здатні використовувати тільки попередній контекст. В розпізнаванні мовлення, де висловлювання записується одразу, доцільно досліджувати і майбутній контекст. Двонаправленні РНМ можуть це робити, обробляючи дані в обох напрямках з двома окремими прихованими шарами, які далі подаються до єдиного вихідного шару.

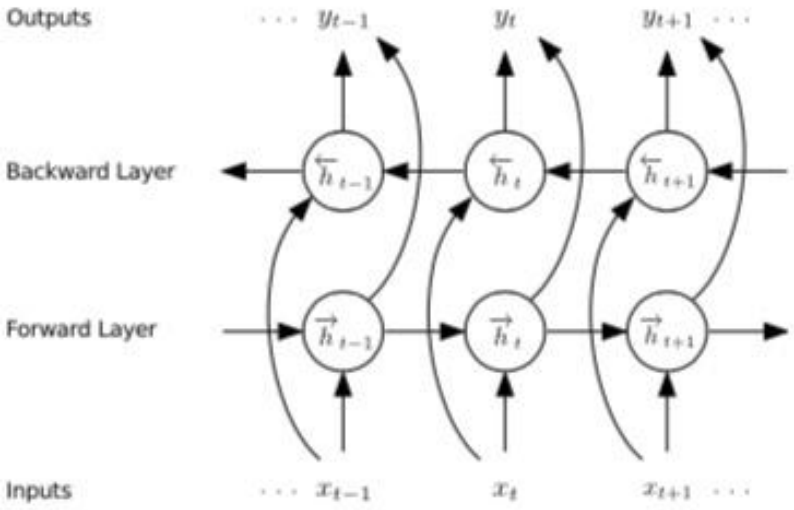


Рисунок 7 – Двонаправлені РНМ

Як показано на рисунку 7, двонаправлені РНМ обчислюють пряму приховану послідовність $\vec{h}$, зворотню приховану послідовність $\overleftarrow{h}$ та вихідну послідовність y, ітеруючи розрахунок зворотнього шару від t = T до 1, та прямого шару від t = 1 до T і покроково оновлюючи вихідний шар:

$$\overrightarrow{h_t} = H(W_{x\vec{h}}\, x_t + W_{\vec{h}\vec{h}}\overrightarrow{h_{t-1}} + b_{\vec{h}} \quad (50)$$

$$\overleftarrow{h_t} = H(W_{x\overleftarrow{h}}\, x_t + W_{\overleftarrow{h}\overleftarrow{h}}\overleftarrow{h_{t+1}} + b_{\overleftarrow{h}} \quad (51)$$

$$y_t = W_{\vec{h}y}\, \overrightarrow{h_t} + W_{\overleftarrow{h}y}\, \overleftarrow{h_t} + b_y \quad (52)$$

Об’єднання двонаправлених РНМ з ДКЧП дає двонаправлену мережу ДКЧП, яка має доступ до довгострокового контексту в обох напрямках вхідних даних.

Розглянемо, як відбувається навчання такої мережі. Нехай у нас є РНМ, в якій будуть зіставлятись звукові послідовності та фонеми. Будемо використовувати виходи мережі для параматеризації розподілу Pr(y|x) використовуючи всі можливі вихідні послідовності фонем з урахуванням вхідної звукової послідовності x. Прологарифмована ймовірність logPr(z|x) цільової вихідної послідовності z може бути змінена з урахуванням ваг мережі використовуючи метод зворотнього поширення в часі (англ.

Backpropagation through time), а вся система може бути оптимізована за допомогою грдієнтного спуску. Опишемо метод для визначення вихідного розподілу, інакше кажучи, навчання мережі. Припускаємо, що розмірність x – T, розмірність z – U, кількість можливих фонем – K.

Цей метод відомий як нейромережева часова класифікація (англ. Connectionist Temporal Classification). Він використовує softmax шар для визначення окремих вихідних ймовірностей Pr(k|t) на кожному кроці t впродовж проходу по вхідній послідовності. Цей розподіл покриває K фонем плюс додатковий пустий символ Ø, який представляє собою тишину (тому розмір softmax шару дорівнює K + 1). Мережа на кожному кроці вирішує, куди відправляти символ. В сумі ці рішення формують розподіл між вхідними даними та цільовими послідовностями. Далі оцінки НЧК використовуються з алгоритмом прямого-оберненого ходу для оновлення вагів нейронної мережі.

Для кращої роботи РНМ додатковим кроком може бути регуляризація – додавання деякої інформації для покращення розв’язку задачі, або навпаки усунення, задля уникнення перенавчання. Для даної задачі це може бути раннє зупинення виконання та ваговий шум (додавання Гаусового шуму до вагів мережі під час навчання, причому лише один раз, а не на кожному часовому кроці).

2.4 Лінгвістичне (мовне) моделювання

Мовна модель дає змогу знайти найбільш ймовірні послідовності слів. Даний процес напряму залежить від конкретної мови. Якщо говорити про англійську мову, то достатньо буде використати N-грамні (статистичні) моделі. Українська мова відноситься до високофлективних мов, тобто таких, що можуть мати багато форм одного і того ж слова. І справді, в англійській мові відсутнє таке явище як відмінки, в той час як в українській мові вони існують практично для всіх слів. В таких випадках, статистичні моделі вже

не є ефективними, оскільки для точного результату словник даних доведеться збільшити у декілька разів. Тому в цих ситуаціях застосовуються гібридні мовні моделі, що використовують лексичні та синтаксичні правила української мови, граматичні характеристики слів і т. д.

2.5 Граматики

Граматика описує простий тип мови для управління і керування. Зазвичай вони написані вручну або генеруються автоматично в межах коду. Граматики зазвичай не містять у собі інформацію про ймовірність для послідовностей слів, але деякі елементи можуть бути мати свою вагу. Вони можуть бути створені за допомогою формату граматики мови Javascript (JSGF – Javascript Grammar File) і зазвичай мають розширення файлу .gram або .jsgf.

Граматики дозволяють точно вказати чіткі правила вхідної промови, наприклад, що одне слово може бути повторене лише два-три рази. Проте, інколи ця суворість може бути шкідливою, якщо користувач не знає правил, які вимагає граматика. У цьому випадку все розпізнавання буде невдалим. Тому краще створювати граматики більш гнучкими. Замість фрази достатньо прописати список слів, які можуть зустрічатись у довільному порядку. Варто уникати складних граматик з багатьма правилами і випадками, так як вони уповільнюють процес розпізнавання.

2.6 Декодування

Кінцевий етап розпізнавання, в якому до вхідного звукового потоку ставляться у відповідність результати, отримані при акустичному та мовному моделюванні та визначається найбільш ймовірна послідовність слів.

2.7 Порівняльний аналіз методів розпізнавання голосу

Тренування акустичної моделі це складний і багатоетапний процес. Тому важливо заранння обрати, який із методів моделювання в якій ситуації варто використовувати.

Візьмемо та порівняємо два класи моделей – генеративні (Приховані Марківські моделі – ПММ) та дискримінативні (Long Short Term Memory рекурентні нейронні мережі). Обидві моделі були детально описані в попередніх розділах.

Приховані марковські моделі (ПММ) значно простіше реалізувати, ніж рекурентні нейронні мережі (РНМ). Однак зазвичай вони покладаються на сильні припущення, які не завжди можуть бути точними. Якщо припущення вірні, то ПММ будуть показувати кращу продуктивність, оскільки вони менш вибагливі для виконання задач. З іншого боку, якщо у нас є велика кількість даних, то РНМ моделі матимуть значну перевагу у вихідних результатах.

Можна сказати, що Марківські ланцюги не мають власної “пам’яті”. Ймовірність наступного символу обчислюється на основі попередніх n символів. На практиці n обмежується невеликим значенням (скажімо, 3–5), оскільки матриця переходу експоненціально зростає. Тому речення, що генеруються даними моделями, дуже непослідовні.

Уявімо ситуацію, коли ви намагаєтесь передбачити кожну хвилину дня, чи людина спить, в залежності від даних руху. Ймовірність переходу від сплячого стану до моменту прокидання, збільшується з кожною хвилиною, якщо людина спить. РНМ моделі можуть використовувати цей зв’язок для більшої точності прогнозування.

Це можна спробувати обійти, наприклад, прийнявши попередній стан як функцію. Але додаткова складність не завжди збільшує точність прогнозування ПММ, і до того ж це, безумовно, впливає на час обчислень.

Для кращого розпізнавання доцільно попередньо визначати загальну кількість станів. Повертаючись до прикладу сну, може виникнути думка, що

ніби є лише два стани, які нас цікавлять. Однак, навіть якщо нам потрібно дізнатись прогноз стану людини (спить чи ні), наша модель може отримати важливу інформацію для навчання з додаткових станів, такі як водіння, душ і т.д. (наприклад, душ зазвичай приймають безпосередньо перед сном). Знову ж таки, РНМ теоретично могла б вивчити таку зв'язки, якщо ми маємо достатню кількість навчальних даних.

З вищевикладеного може скластись враження, що РНМ завжди працюють краще. Але насправді, при невеликому наборі даних та недовгих послідовностях РНМ моделям важко дати точний результат, до того ж вони працюють значно довше.

Отже, для невеликих вибірок доцільно використовувати Приховані Марківські моделі. Тому для даної роботи я обрала саме їх.

2.8 Критерії якості роботи системи розпізнавання голосу укр мови.

World Error Rate (WER) є загальноприйнятою метрикою ефективності системи розпізнавання мовлення.

Метод визначення показника WER полягає у вирівнюванні двох текстових рядків – першим є результат розпізнавання, а другим саме той рядок, який мав бути сказаний. Дане вирівнювання реалізовується за допомогою алгоритму динамічного програмування з обчисленням відстані Левенштейна. Відстань Левенштейна представляє собою “вартість” редагування даних (мініальна кількість або зважена сума операцій редагування) для перетворення першого рядка у другий з найменшою кількістю операцій заміни (S), видалення (D) та вставки (I) слів:

$$WER = (S + D + I) / N \quad (53)$$

де N – кількість слів в розпізнаній фразі.

Для оцінювання результатів автоматичного розпізнавання мовлення використовується показник, як відсоток коректно розпізнаних слів (WCR – Word Correctly Recognized), який не враховує помилкові вставки слів, отриманих системою:

$$WCR = H/T * 100\%, H = N – D – S \quad (54)$$

де H – кількість правильно розпізнаних слів.

2.9 Висновки

У цьому розділі був покроково описаний процес розпізнавання мовлення. Було розглянуто алгоритми та методи, що застосовуються при виділенні ознак зі звукового потоку (MFCC), побудові акустичної (Dynamic Time Wrapping, Приховані Марківські моделі, Рекурентні нейронні мережі) та лінгвістичної моделі (n-грамна мовна модель), граматик (JSGF).

Було виконано порівняння Прихованих Марківських моделей та Рекурентних нейронних мереж для побудови акустичної моделі та обрано найкращий метод для застосування у даній роботі.

Наприкінці були описані універсальні критерії якості роботи системи розпізнавання, якими користуються у всьому світі – World Error Rate та Word Correctly Recognized.

3 АНАЛІЗ АРХІТЕКТУРИ СИСТЕМИ РОЗПІЗНАВАННЯ УКРАЇНСЬКОЇ МОВИ

3.1 Обґрунтування вибору платформи та мови реалізації

Основною платформою для реалізації програмного продукту була обрана операційна система macOS. Це сучасна POSIX-сумісна операційна система, створена компанією Apple, яка швидко розвивається та набирає свою популярність з кожним роком. На жаль, кількість програм, які пишуться під дану ОС значно уступає тому пласту розробок, які зараз існують на більш популярних ОС Windows та Linux. Тому було вирішено створити продукт саме з використанням цієї ОС, так як програм для голосового управління системою дуже обмаль, а аналогів з розпізнаванням української мови немає взагалі.

Для реалізації програмного продукту була обрана мова програмування Python, версія 3.7. Це інтерпретована мова програмування високого рівня з об’єктно-орієнтованим підходом та строгою динамічною типізацією [18]. Ефективні структури даних високого рівня поруч з динамічною семантикою дозволяють швидко та зручно розробляти прикладні програми та скрипти. Серед основних її переваг можна назвати такі:

- чистий та зрозумілий синтаксис;
- багатоплатформність – можливість запуску програм на різних ОС;
- у стандартному дистрибутиві міститься велика кількість корисних та потрібних модулів;
- вбудована підтримка Unicode (це дозволить без проблем працювати з українською мовою)
- зручний для математичних розрахунків;
- вбудований в Mac OS.

Однією з головних причин вибору саме цієї мови стала підтримка великої кількості бібліотек та зручний доступ до них. Розглянемо ті, які будуть використані в подальшій розробці нашого програмного продукту.

Для обчислення різного роду математичних функцій та швидких розрахунків з багатовимірними матрицями та масивами буде використана бібліотека NumPy. Її реалізація дозволяє в рази швидше робити математичні операції у порівнянні з вбудованими в Python функціями.

Модуль librosa призначений для аналізу аудіо-даних. Цей модуль буде корисним для виділення ознак (в тому числі і MFCC-векторів).

Основний пакет, який допоможе нам розпізнавати мовлення має назву pocketsphinx. Він надає доступ Python-інтерфейсу до бібліотеки CMU Sphinxbase, створеної з допомогою програмного забезпечення SWIG та Setuptools. Тож останні два перечислені продукти також необхідно інсталювати аби pocketsphinx міг коректно працювати.

os – модуль, що містить в собі функції для роботи з операційною системою. За допомогою нього, наша програма зможе не лише розпізнавати вхідні команди, сказані користувачем, але й і виконувати їх.

re – модуль для роботи з регулярними виразами. Це такі шаблони для пошуку з допомогою спеціального синтаксису. Функції цього модулю будуть використані для швидкої обробки тексту.

3.2 Аналіз вимог користувача до програмного продукту

Для того, щоб користувач міг без проблем скористатись системою голосового управління ОС, вона має задовольняти деяким базовим критеріям.

По-перше, система має розпізнавати послідовну, злитну мову, тобто таку, якою люди спілкуються у повсякденному житті. Ця задача значно складніша за просте розпізнавання окремих слів.

По-друге програма має бути оптимізована і швидко працювати. Текст, сказаний людиною має практично одразу (1-3 секунди) відображатись на

екрані, і в той же час виконуватись відповідна системна команда. Бо основна суть голосового управління – автоматизоване виконання завдань, яке є швидшим за ручне.

Остання та одна з головних вимог – це універсальність. Програма має розпізнавати будь-який людський голос, незалежно від тембру, статі, віку, інтонації чи швидкості вимови.

Таким чином, можна узагальнити вищесказані вимоги, та зробити висновок про те, що наш програмний застосунок має швидко та точно обробляти голосові записи, приймати рішення про те, які команди були сказані, та миттєво їх виконувати.

3.3 Аналіз архітектури системи розпізнавання голосу

Програмний продукт можна представити у вигляді п’яти модулів:

- assistant.py – виконавчий скрипт, основна частина програми. У ньому налаштовуються параметри для запису звуку та безпосередньо відбувається саме записування. Далі отриманий файл передається як аргумент модулю для виділення ознак, а далі очікується файл з розпізнаним текстом. Скриптом відбувається обробка цього файлу та виклики системних команд.
- mfcc.py – модуль, призначений для виділення ознак зі звукового потоку. Ділить аудіо на фрейми, для кожного фрейму повертає представлення у вигляді mfcc-вектору з розміром 13 для подальшої обробки.
- model.py – модуль для визначення відповідностей фреймам фонем. Використовує акустичну модель з бібліотеки pocketsphinx. Повертає розподіл ймовірностей фонем для кожного фрейму.
- to_text.py – модуль для співставлення послідовності фонем словам та пошуку найбільш підходящої послідовності слів. Використовуються JSGF-граматики та функції бібліотеки

pocketsphinx. Повертається послідовність слів у текстовому вигляді, кінцевий етап розпізнавання.

– g2p.py – модуль для конвертації слів у послідовність фонем. Використовується на підготовчому етапі, для створення словника для розпізнавання.

Функціональна схема, що показує послідовність кроків виконання програми, зображена на рис. 8:

Рис. 8 – Функціональна схема програмного продукту

3.4 Керівництво користувача

Для користування програмою, користувач повинен відкрити Термінал та перейти в папку з програмою. Далі він може запустити прогаму, виконавши команду: python3 assistant.py. Його запит має бути чітко сформульований. Спершу в ньому має йти привітання (“Привіт, комп’ютер” або “Окей, комп’ютер”). Далі – безпосередньо сама команда (“відкритй програму Калькулятор”, “покажи папку Робочий стіл”). Дані правила запитів задаються JSGF-граматикою, яка описана у файлі ukr.jsgf, скріншот якого зображено на рис. 8:

ukr.jsgf

```
#JSGF V1.0;

grammar ukr_calc;

<greeting> = (привіт|ок|окей)(комп'ютер);
<action> = (відкрий|включи|виключи|закрий|запусти|згорни|покажи|
сховай|розгорни);
<object> = (папку|програму);
<name> = (
Інтернет|Калькулятор|Пошта|Фото|Шахмати|Книги|Календар);
<expression> = <action> <object> <name>;
public <query> = <greeting> <expression>;
```

Рис. 8 – Файл ukr.jsgf

Після цього програма слухає сказазаний користувачем запит. В будь-який момент запис можна зупинити, натиснувши комбінацію клавіш Ctrl+C. Після зупинки запису, програма одразу починає обробку записаного файлу та через декілька секунд видає результат розпізнавання та виконує команду, сказану користувачем (рис. 9).

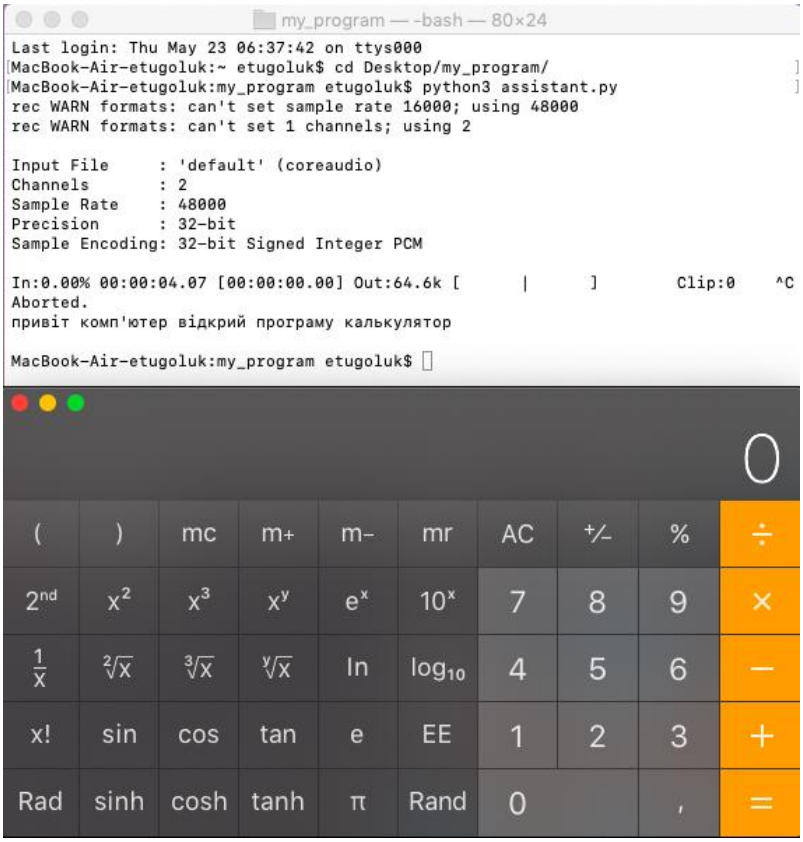


Рис. 9 – Робота програми

Далі програма закінчує виконання.

3.5 Аналіз результатів, отриманих в роботі

3.5.1 Опис та аналіз вибірки вхідних даних

Для швидкої та оптимізованої роботи системи розпізнавання використовується не цілий словник української мови, а лише його невелика частина (приблизно 100 слів). Даний словник представлений у файлі ukr_vocab.txt, а його конвертований вигляд у послідовність фонем зберігається у файлі g2p_ukr.dict (рис. 10).

```
виключи v yy k ll uj ch i
відкрий v i d k r y j
вісім v i ss ii mm
включи v k ll ju ch i
два d v aa
девять dd ee vv j ja tt
десять dd ee ss ja tt
закрий z a k r yy j
запусти z ay p u ss tt y
згорни z g oo r n y
Інтернет i nn tt i r nn je t
Календар k a l ee n d aa r
Калькулятор k ay ll k u ll ja t ay r
Книги k n y g y
комп'ютер k oo m pp j ju tt i r
мінус mm ii n u s
нуль n u ll
один oo dd yy n
ок oo k
окей oo k ee j
```

Рис. 10 – Словник для розпізнавання

Очевидно, що у словнику містяться не випадково обрані слова, а саме ті, які необхідні для управління операційною системою. Їх можна поділити на такі групи:

- привітання («привіт», «ок», «окей»)
- дії («відкрий», «включи», «виключи», «закрий», «згорни», «покажи» і т.д)
- об’єкти («програма», «папка», «комп’ютер»)
- список назв програм («Калькулятор», «Пошта» і т.д)

3.5.2 Дослідження практичних результатів

Для перевірки якості результатів програми, вона була протестована за критеріями відстань, шум та швидкість вимови. Також для більш точного результату, програма тестувалась декількома користувачами різного статі, віку та з різним тембром голосу (троє жінок віком від 20 до 53 років та троє чоловіків від 14 до 31 року).

Усереднені результати тестування наведені в таблицях 1 – 3:

Таблиця 1 – Відсоток коректно розпізнаних слів за критеріями відстані та шуму

Відстань \ Шум	тихо	шумно
біля комп'ютера	100%	80%
1 м	100%	80%
3 м	80%	60%

Таблиця 2 – Відсоток коректно розпізнаних слів за критеріями швидкості та шуму

Швидкість \ Шум	тихо	шумно
повільно	100%	80%
нормально	100%	80%
швидко	80%	60%

Таблиця 3 – Відсоток коректно розпізнаних слів за критеріями швидкості та відстані

Швидкість \ Відстань	біля комп’ютера	1 м	3 м
повільно	100%	100%	80%
нормально	100%	100%	80%
швидко	80%	80%	60%

В цілому програма показала гарний результат на практиці. Їй вдається розпізнавати голоси людей різного тембру, статі та віку. Програма видає

стовідсоткову точність результатів, якщо більш роздільно промовляти слова. Але навіть і при швидкому злитному потоці слів маємо WER 80% (може бути не розпізнано одне слово зі всього речення). Ще така ситуація може траплятись у випадку посторонніх шумів та багатоголосся, коефіцієнт розпізнавання може падати до 60% або два нерозпізнаних слова, але це загальна проблема більшості розпізнавачів на сьогоднішній день.

3.6 Висновки

У третьому розділі було детально розглянуто архітектуру програмного продукту. Для реалізації застосунку була обрана операційна система Mac OS та мова програмування Python. Був проведений детальний опис їх переваг, а також бібліотек та модулів мови Python, які були застосовані при розробці програми.

Були визначені основні модулі, які мають бути створенні для повноцінної реалізації поставленої задачі. Розроблено детальне керівництво користувача для зручної роботи з програмою.

Для кращого представлення програми, наступним пунктом був опис вхідної вибірки даних, а саме словника для розпізнавання. Це дало краще розуміння того, які слова необхідні казати, аби програма могла виконувати команди ОС.

Останнім і найголовнішим завданням цього розділу було проведення аналізу результатів роботи програми.

4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

4.1 Постановка задачі техніко-економічного аналізу

У даному розділі проводиться оцінка основних характеристик програмного продукту, призначеного для локального голосового управління персональним комп'ютером з вбудованою системою розпізнавання української мови. Програмний продукт призначено для використання на персональних комп'ютерах під управлінням операційної системи Mac OS. Інтерфейс користувача був розроблений за допомогою мови програмування Python у середовищі розробки PyCharm.

Нижче наведено аналіз різних варіантів реалізації модулю з метою вибору оптимальної, з огляду при цьому як на економічні фактори, так і на характеристики продукту, що впливають на продуктивність роботи і на його сумісність з апаратним забезпеченням. Для цього було використано апарат функціонально-вартісного аналізу.

4.1.1 Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка програмного продукту, який вирішує задачу розпізнавання української мови та виконує відповідні голосові команди, сказані людиною.

Виходячи з конкретної мети, можна виділити наступні основні функції програми:

F_1 – вибір мови програмування;

F_2 – вибір бібліотеки для розпізнавання;

F_3 – вибір середовища розробки.

Кожна з основних функцій може мати декілька варіантів реалізації.

Функція F_1:

а) Python;

б) C++;

Функція F_2:

а) Pocketsphinx;

б) Julius;

Функція F_3:

а) PyCharm;

б) Visual Studio.

4.1.2 Варіанти реалізації основних функцій

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 11). На основі цієї карти побудовано позитивно-негативну матрицю варіантів основних функцій (табл. 4).

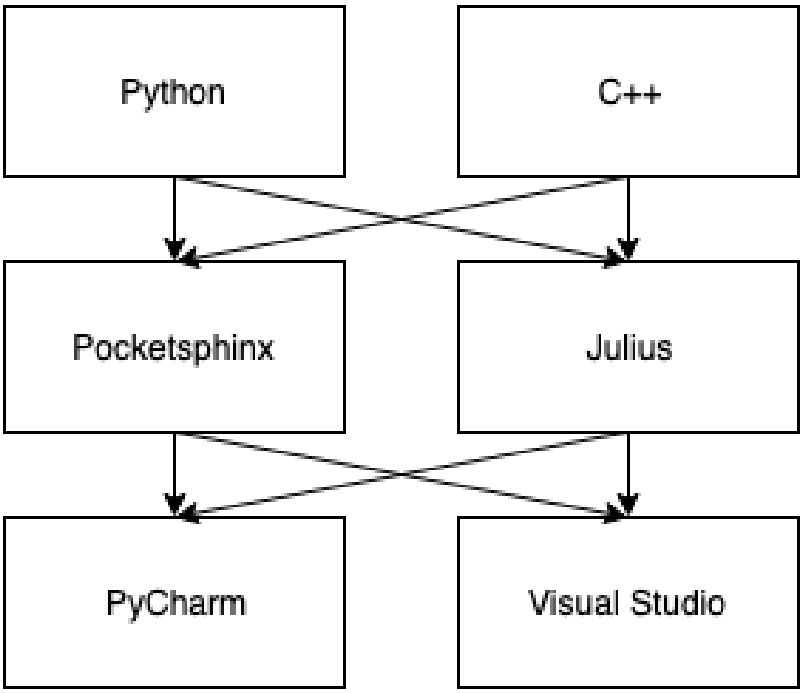


Рисунок 11 – Морфологічна карта

Морфологічна карта відображує всі можливі комбінації варіантів реалізації функцій, які складають повну множину варіантів ПП.

Таблиця 4 – Позитивно-негативна матриця

Основні функції	Варіанти реалізації	Переваги	Недоліки
F1	А	Кросплатформений, Займає менше часу при написанні коду	Низька швидкодія, більший час на виконання операцій
	Б	Висока швидкодія, оптимізація пам'яті	Займає більше часу при написанні коду
F2	А	Безкоштовність, чудова документація, швидкість	Відсутність українського словника та акустичної моделі
	Б	Надійність, безкоштовність	Відсутність українського словника та акустичної моделі, повільний
F3	А	“Розумний” редактор коду, зручна навігація, швидкий та безпечний рефакторинг	Повільний
	Б	Висока інтегрованість додаткових інструментів	Займає дуже багато пам'яті

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція *F1:*

Оскільки поставлена задача є багатоетапною та достатньо складною, доцільно обрати мову, яка спростить процес написання коду та зекономить час. Для цього підійде мова програмування Python – варіант А.

Функція *F2:*

Обидві бібліотеки для розпізнавання мають хороший функкціонал, який підходить для розв’язання поставленої задачі. Тому можемо розглядати обидва варіанти А та Б.

Функція *F3:*

Оскільки, програмний продукт реалізується мовою Python, використовуємо варіант А, так як дане середовище розробки створене саме для цієї мови.

Таким чином, будемо розглядати такі варіанти реалізації ПП:

1. F1а – F2а – F3а
2. F1а – F2б – F3а

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

4.2 Обґрунтування системи параметрів ПП

4.2.1 Опис параметрів

На підставі даних про основні функції, що повинен реалізувати програмний продукт, вимог до нього, визначаються основні параметри виробу, що будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

- X1 – швидкодія мови програмування;
- X2 – об’єм пам’яті для збереження даних;
- X3 – час обробки даних;
- X4 – потенційний об’єм програмного коду.

X1: Відображає швидкодію операцій мови програмування залежно від обраної мови програмування.

X2: Відображає об’єм пам’яті в оперативній пам’яті персонального комп’ютера, необхідний для збереження та обробки даних під час виконання програми.

X3: Відображає час, який витрачається на дії.

X4: Показує розмір програмного коду який необхідно створити безпосередньо розробнику.

4.2.2 Кількісна оцінка параметрів

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію ПП як показано у таблиці 5.

Таблиця 5 – Основні параметри ПП

Назва Параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія мови програмування	X1	с	15	10	3
Об’єм пам’яті для збереження даних	X2	Кб	1000	300	100
Час обробки запитів користувача	X3	с	20	3	1
Потенційний об’єм програмного коду	X4	кількість рядків коду	3000	1000	500

За даними, наведеними у таблиці 5, будуються графічні характеристики параметрів – рис. 12-15.

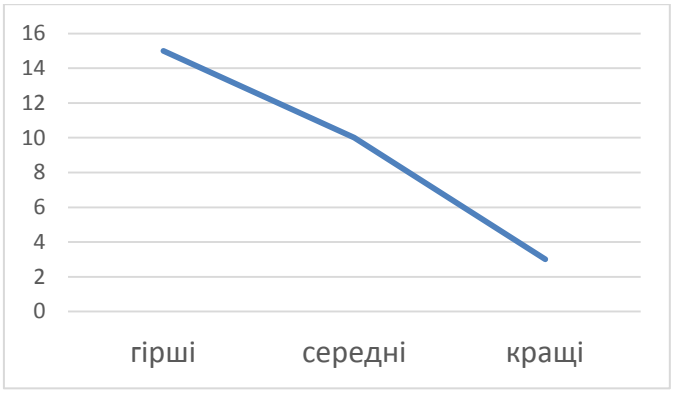


Рисунок 12 – X1, час ознайомлення з мовою програмування

Рисунок 13 – X2, об’єм пам’яті для збереження даних

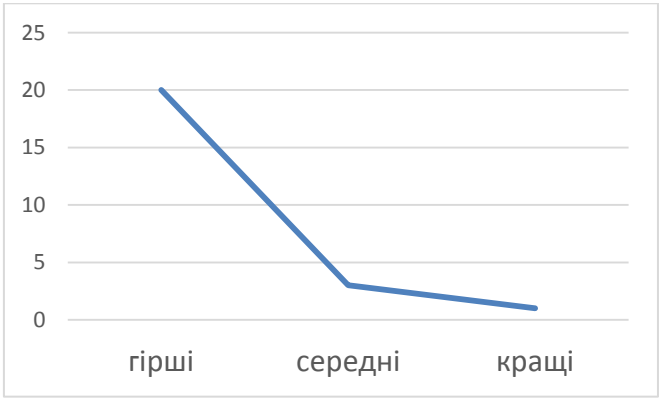


Рисунок 14 – X3, час обробки даних алгоритмом

Рисунок 15 – X4, потенційний об’єм програмного коду

4.2.3 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який має найбільш зручний інтерфейс та зрозумілу взаємодію з користувачем.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

- визначення рівня значимості параметра шляхом присвоєння різних рангів;
- перевірку придатності експертних оцінок для подальшого використання;
- визначення оцінки попарного пріоритету параметрів;
- обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 6.

Таблиця 6 – Результати ранжування параметрів

Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δi	Δi^2
		1	2	3	4	5	6	7			
Швидкодія мови програмування	с	1	2	2	1	2	1	2	11	-6,5	42,25
Об’єм пам’яті для збереження даних	Кб	2	1	1	2	1	2	1	10	-7,5	56,25
Час обробки запитів користувача	с	4	4	3	4	4	4	3	26	8,5	72,25
Потенційний об’єм	кількість	3	3	4	3	3	3	4	23	5,5	30,25

програмного коду	рядків коду										
Разом		10	10	10	10	10	10	10	70	0	200

Для перевірки ступені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70$$

де N – число експертів, n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17{,}5$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T.$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г)загальна сума квадратів відхилення:

$$S = 200$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 * 200}{7^2(4^3 - 4)} = 0{,}816 > W_k = 0{,}67$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 7.

Таблиця 7 – Попарне порівняння параметрів

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	<	>	>	<	>	<	>	>	1.5
X1 і X3	<	<	<	<	<	<	<	<	0.5
X1 і X4	<	<	<	<	<	<	<	<	0.5
X2 і X3	<	<	<	<	<	<	<	<	0.5
X2 і X4	<	<	<	<	<	<	<	<	0.5
X3 і X4	>	>	<	>	>	>	<	>	1.5

Числове значення, що визначає ступінь переваги і–го параметра над j–тим, аіj визначається по формулі:

$$a = \begin{cases} 1{,}5 \text{ при } X_i > X_j \\ 1 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases}$$

З отриманих числових оцінок переваги складемо матрицю $A = \| a_{ij} \|$. Для кожного параметра зробимо розрахунок вагомості $K_{ві}$ за наступними формулами:

$$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i}, \text{де } b_i = \sum_{i=1}^{N} a_{ij}$$

Відносні оцінки розраховуються декілька разів доти, поки наступні

значення не будуть незначно відрізнятися від попередніх (менше 2%).На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{\text{ві}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \text{де } b_i' = \sum_{i=1}^{N} a_{ij} b_j$$

Як видно з таблиці 8, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 8 – Розрахунок вагомості параметрів

Параметри	Параметри x_j				Перша ітер.		Друга ітер.		Третя ітер	
	X1	X2	X3	X4	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$	b_i^2	$K_{\text{ві}}^2$
X1	1.0	1.5	0.5	0.5	3.5	0.2186	12.25	0.209	44.875	0.21
X2	0.5	1.0	0.5	0.5	2.5	0.1562	9.25	0.156	34.125	0.16
X3	1.5	1.5	1.0	1.5	5.5	0.344	21.25	0.36	76.25	0.358
X4	1.5	1.5	0.5	1.0	4.5	0.2812	16.25	0.275	58	0.272
Всього:					16	1	59	1	213,25	1

4.3 Аналіз рівня якості варіантів реалізації функцій

Визначимо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів X1 (швидкодія мови програмування), X2 (об’єм пам’яті для збереження даних) та X4 (потенційний об’єм програмного коду) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра X3 (час обробки даних) обрано не найгіршим (не максимальним), тобто це значення відповідає або варіанту а) 1 с або варіанту б) 3 с.

Коефіцієнт технічного рівня для кожного варіанта реалізації програми розраховується так (таблиця 9):

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j}$$

де n – кількість параметрів; в – коефіцієнт вагомості і–го параметра; B_i – оцінка і–го параметра в балах.

Таблиця 9 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1(X1)	А	10	5	0.21	1.05
F2(X3)	А	1	4	0.358	1.432
	Б	3	2	0.358	0.716
F2(X4)	А	1000	5	0.272	1.36
F3(X2)	А	300	5	0.16	0.8

За даними з таблиці К за формулою

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}]$$

визначаємо рівень якості кожного з варіантів:

K_{K1} = 1.05 + 1.432 + 1.36 + 0.8 = 4.642

K_{K2} = 1.05 + 0.716 + 1.36 + 0.8 = 3.926

Як видно з розрахунків, кращим є перший варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.4 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка акустичної моделі та системи розпізнавання української мови;
2. Розробка програмної оболонки.

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування кожного із завдань.

Загальна трудомісткість обчислюється як

$$T_O = T_P * K_П * K_{СК} * K_М * K_{СТ} * K_{СТ.М},$$

де T_P – трудомісткість розробки ПП; $K_П$ – поправочний коефіцієнт; $K_{СК}$ – коефіцієнт на складність вхідної інформації; $K_М$ – коефіцієнт рівня мови програмування; $K_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм; $K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення.

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: T_P = 90 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $K_П$ = 1.7.

Поправочний коефіцієнт, який враховує складність контролю вхідної та

вихідної інформації рівний 1: $К_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{СТ}$ = 0.8. Тоді, за формулою 5.1, загальна трудомісткість програмування першого завдання дорівнює:

$$Т_1 = 90 * 1.7 * 0.8 = 122.4 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $Т_Р$ = 27 людино-днів, $К_П$ = 0.9, $К_{СК}$ = 1, $К_{СТ}$ =0.8:

$$Т_2 = 27 * 0.9 * 0.8 = 19.44 \text{ людино-днів.}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$$Т_I = (122.4 + 19.44 + 4.642 + 19.44)\ 8 = 1327{,}326 \text{ людино-годин;}$$

$$Т_{II} = (122.4 + 19.44 + 3.926 + 19.44)\ 8 = 1321.648 \text{ людино-годин;}$$

Більш високу трудомісткість має варіант І.

В розробці беруть участь два програмісти з окладом 10000 грн. Визначимо зарплату за годину за формулою:

$$Сч = \frac{М}{Т_m \cdot t} \text{ грн.,}$$

де М – місячний оклад працівників; T_m – кількість робочих днів тиждень; t – кількість робочих годин в день.

$$С_ч = \frac{10000 + 10000}{2 * 21 * 8} = 59.5 \text{ грн.}$$

Тоді, розрахуємо заробітну плату за формулою:

$$С_{зп} = С_ч * T_i * К_д,$$

де $С_ч$ – величина погодинної оплати праці програміста; $Т_і$ – трудомісткість відповідного завдання; $К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП}$ = 59.5 * 1327,326 * 1.2 = 94771,08 грн.

II. $С_{ЗП}$ = 59.5 * 1321.648 * 1.2 = 94365,67 грн.

Відрахування на єдиний соціальний внесок становить 22%:

I. $С_{ВІД} = С_{ЗП} * 0.22$ = 94771,08 * 0.22 = 20849.64 грн.

II. $С_{ВІД} = С_{ЗП}$ * 0.22 = 94365,67 * 0.22 = 20760.45 грн.

Визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного програміста з окладом 10000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$С_Г$ = 12 * М * $К_З$ = 12 * 10000 * 0,2 = 24000 грн.

З урахуванням додаткової заробітної плати:

$С_{ЗП} = С_Г$ * (1+ $К_З$) = 24000 * (1 + 0.2) = 28800 грн.

Відрахування на єдиний соціальний внесок:

$С_{ВІД} = С_{ЗП}$ * 0.22= 28800 * 0,22 = 6336 грн.

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 25000 грн.

$С_А = К_{ТМ} * К_А * Ц_{ПР}$ = 1.15 * 0.25 * 25000 = 7187.5 грн.,

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача; $К_А$– річна норма амортизації; $Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$С_Р = К_{ТМ} * Ц_{ПР} * К_Р$ = 1.15 * 25000 * 0.05 = 1437.5 грн.,

де $К_Р$ – відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$Т_{ЕФ} = (Д_К – (Д_В + Д_С) – Д_Р) t_З К_В$ = (365 – 116 – 4) * 8 * 0.9 = 1764 годин,

де $Д_К$ – календарна кількість днів у році; $Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів; $Д_Р$ – кількість днів планових ремонтів

устаткування; t – кількість робочих годин в день; $К_В$ – коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$С_{ЕЛ} = Т_{ЕФ} * N_С * Ц_{ЕН}$ =1764 * 0,156 * 2,7515 = 757.168 грн.,

де $N_С$ – середньо-споживча потужність приладу; $Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$С_Н = Ц_{ПР} * 0.67$ = 25000 * 0,67 = 16750 грн.

Тоді, річні експлуатаційні витрати будуть:

$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н$

$С_{ЕКС}$ = 28800 + 6336 + 7187.7 + 1437.5 + 757.17 + 16750 = 61268.37 грн.

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$С_{М-Г} = С_{ЕКС} / Т_{ЕФ}$ = 61268.37 / 1764 = 34.73 грн/час.

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$С_М = С_{М-Г} * Т$

I. $С_М$ = 34.73 * 1327,326 = 46098.032 грн.;

II. $С_М$ = 34.73 * 1321.648 = 45900.835 грн.;

Накладні витрати складають 67% від заробітної плати: $С_Н = С_{ЗП} * 0,67$

I. $С_Н$ = 94771,08 * 0,67 = 63496.624 грн.;

II. $С_Н$ = 94365,67 * 0,67 = 63225 грн.;

Отже, вартість розробки ПП за варіантами становить:

$С_{ПП} = С_{ЗП} + С_{ВІД} + С_М + С_Н$

I. $С_{ПП}$ = 94771,08 + 20849.64 + 46098.032 + 63496.624 = 225215.376 грн.;

II. $С_{ПП}$ = 94365,67 + 20760.45 + 45900.835 + 63225 = 224251.955 грн.;

4.5 Вибір кращого варіанта ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕРj} = К_{Кj} / С_{Фj},\ К_{ТЕР1} = 4.642/\ 225215.376\ = 20{,}61 * 10^{-6};$$

$$К_{ТЕР2} = 3.926 / 224251.955 = 17{,}51 * 10^{-6};$$

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1}= 20{,}61 *10^{-6}$.

4.6 Висновки

В даному розділі проведено повний функціонально-вартісний аналіз системи розпізнавання мовлення, яка була розроблена в рамках дипломного проекту. Процес аналізу можна умовно розділити на дві частини.

У першій частині було проведено дослідження ПП з технічної точки зору: визначено основні функції ПП та сформовано варіанти їх реалізації; на основі обчислених значень параметрів, а також експертних оцінок щодо їх важливості було обчислено коефіцієнт технічного рівня, з допомогою якого було визначено оптимальний варіант реалізації функцій ПП з технічної точки зору.

У другій частині варіанти реалізації були обґрунтовані з економічної точки зору. Порівняння запропонованих варіантів реалізації в рамках даної частини виконувалось за коефіцієнтом ефективності, для обчислення якого були обчислені такі допоміжні параметри, як трудомісткість, витрати на заробітну плату, накладні витрати.

Після виконання функціонально-вартісного аналізу ПП, було виявлено, що оптимальним є перший варіант реалізації з показником техніко-економічного рівня якості $К_{ТЕР} = 20{,}61 *10^{-6}$.

Обраний варіант реалізації програмного продукту має такі параметри:

- мова програмування – Python;
- бібліотека для розпізнавання – Pocketsphinx;
- середовище розробки – PyCharm.

Даний варіант виконання програмного продукту відповідає усім поставленим вимогам, та може бути розроблений відносно швидко.

ВИСНОВКИ

Основною задачею дипломної роботи була розробка системи розпізнавання української мови. В роботі були отримані наступні результати:

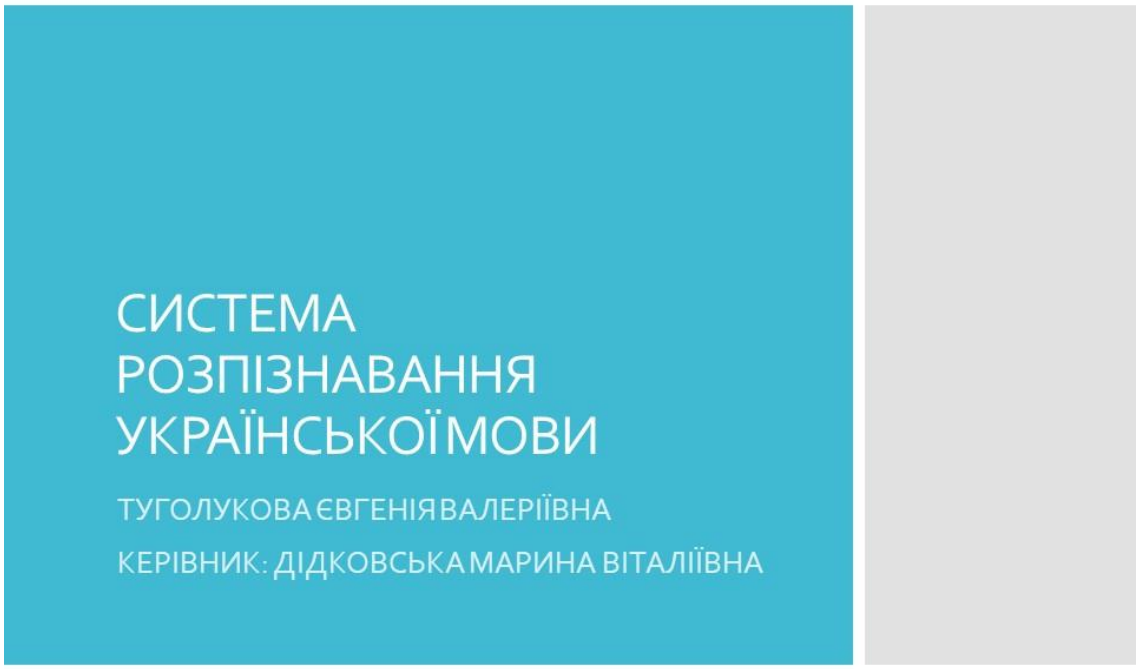
- проведено дослідження сучасної ситуації у сфері розпізнавання, особливо, що стосується слов'янських мов. Розглянуто існуючі платформи, виявлено основні проблеми та задачі, що стоять перед розробниками даних систем.
- детально розглянуто та математично описано процес розпізнавання акустичної інформації та виділено основні етапи, а саме: запис та оцифрування звукових даних, попередня обробка та виділення ознак, робота акустичної моделі для співставлення звуковій послідовності фонемної, робота лінгвістичної моделі для знаходження ймовірності послідовності слів, побудова граматичного словника та декодування. Проаналізовано переваги і недоліки Прихованих Марківських моделей та Рекурентних нейронних мереж для побудови акустичної моделі. Описано універсальні критерії якості роботи системи розпізнавання.
- У практичній частині роботи був обраний метод розпізнавання на основі Прихованих Марківських моделей із застосуванням граматик на базі JSGF. Була побудована архітектура програмного продукту з модулями запису та обробки голосових команд, конвертації слів у послідовність фонем, обчисленням MFCC-векторів, моделюванням та співставленням звуковим фреймам фонем, обробка JSGF-граматики.
- На базі створеної архітектури була розроблена система голосового управління на базі операційної системи Mac OS, яка розпізнає фрази, сказані українською мовою та виконує відповідні команди.

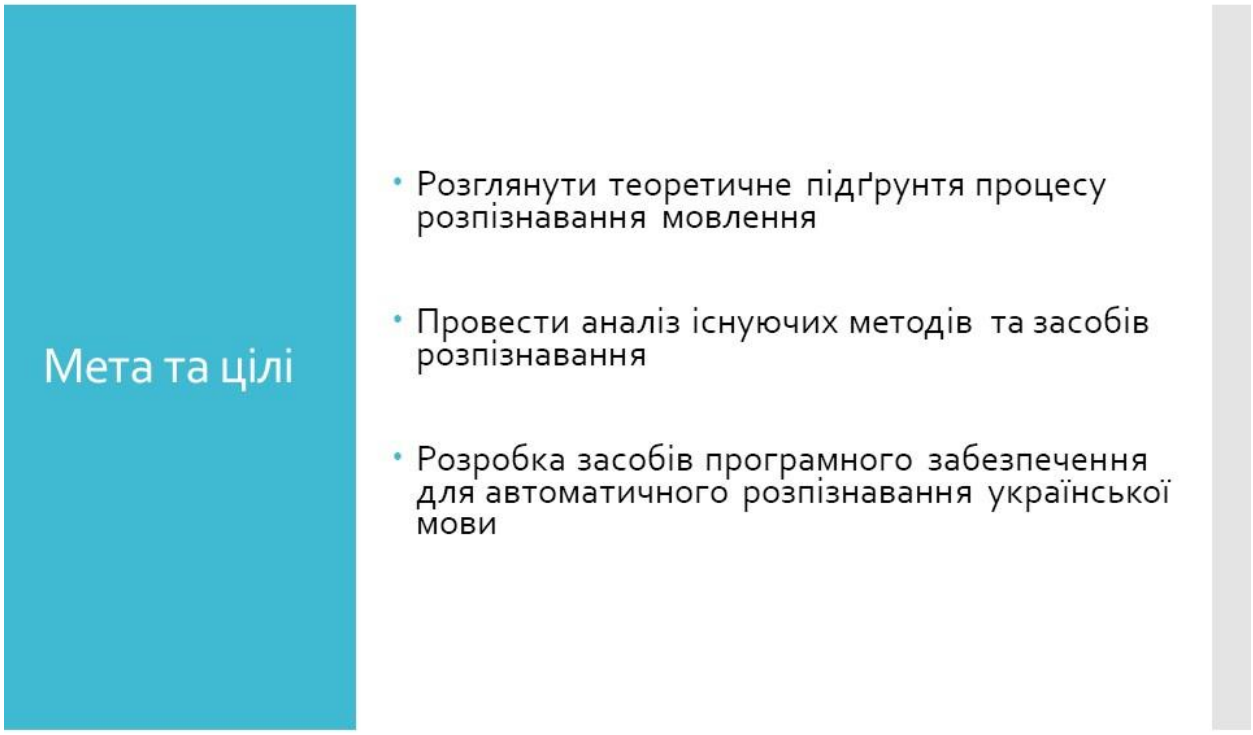
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Robeiko V., Sazhok M. Real-time spontaneous Ukrainian speech recognition system based on word acoustic composite models. URL: https://pdfs.semanticscholar.org/7cf7/a97ae62ba46af0525cf5b5eb8905da4910f7.pdf
2. Boyd C. The Past, Present, and Future of Speech Recognition Technology. URL: https://medium.com/swlh/the-past-present-and-future-of-speech-recognition-technology-cf13c179aaf
3. Saksamudre S., Shrishrimal P., Deshmukh R. A Review on Different Approaches for Speech Recognition System. URL: https://pdfs.semanticscholar.org/b909/3377c6579b97ab8bd5d4dd9947d372dddc2e.pdf
4. Matarneh R., Maksymova S., Lyashenko V., Belova N. Speech Recognition Systems: A Comparative Review. URL: https://pdfs.semanticscholar.org/8c3b/5bab98556f57dbc8142d5f3f8ad13109c733.pdf
5. Otander J. CMUSphinx tutorial for developers. URL: https://cmusphinx.github.io/wiki/tutorial/
6. Hannun A. Speech Recognition Is Not Solved. URL: https://awni.github.io/speech-recognition/
7. Иванов В.И., Тимофеев М.В. Идентификация голоса человека на основе мел-частотных кепстральных коэффициентов. Молодежный научный форум: Технические и математические науки. С. 1–2.
8. Алборова Ж. В. Алгоритмы и методы распознавания речи. Молодежный научно-технический весник. 9 чер. 2016. С. 4 – 7.
9. Малютин А., Шатохин П. Применение алгоритма динамического трасформирования для распознавания отдельных слов в ограниченном словаре. URL: http://mkg.donntu.edu.ua/wp-content/uploads/2017/09/mkg_13.pdf

10. Rabiner L. R. A tutorial on Hidden Markov models and selected applications in speech recognition. Proceedings of the IEEE, 1989, vol. 77, no. 2, pp. 257–286. Doi: 10.1109/5.18626
11. Degirmenci A. Introduction to hidden markov models. URL: https: //scholar.harvard.edu/files/adegirmenci/files/hmm_adegirmenci_2014.pdf
12. Graves A., Mohamed A., Hinton G. Speech recognition with deep recurrent neural networks. URL: https://arxiv.org/pdf/1303.5778.pdf
13. Olah C. Understanding LSTM networks. URL: https://colah.github.io /posts/2015-08-understanding-lstms/
14. Karpathy A. Understanding the unreasonable effectiveness of recurrent neural networks. URL: http://karpathy.github.io/2015/05/21/rnn-effectiveness/
15. Hunt A. JSpeech grammar format. URL: https://www.w3.org/tr /2000/note-jsgf-20000605/
16. Panzner M., Cimiano P. Comparing Hidden Markov models and Long Short Term memory neural networks for learning action representations. URL:https://pub.unibielefeld.de/download/2903474/27910/MOD2016.pdf
17. Карпов А., Кипяткова И. Методология оценивания работы систем автоматического распознавания речи. URL: http://pribor.ifmo.ru /file/article/5994.pdf
18. Guido van Rossum, Python Reference Manual, release 2.4.4, 18 October 2006

ДОДАТОК А


СИСТЕМА
РОЗПІЗНАВАННЯ
УКРАЇНСЬКОЇМОВИ
ТУГОЛУКОВА ЄВГЕНІЯ ВАЛЕРІЇВНА
КЕРІВНИК: ДІДКОВСЬКА МАРИНА ВІТАЛІЇВНА


Мета та цілі
Розглянути теоретичне підґрунтя процесу розпізнавання мовлення
Провести аналіз існуючих методів та засобів розпізнавання
Розробка засобів програмного забезпечення для автоматичного розпізнавання української мови

Актуальність

- Простота та швидкість користування такими системами
- Доступність мовного інтерфейсу людям з порушеннями опорно-рухового та зорового апарату
- Невелика кількість подібних розробок для слов'янських мов

Засоби для розробки

Операційна система	Mac OS
Мова програмування	Python
Середовище розробки	PyCharm
Бібліотека для розпізнавання	Pocketsphinx

Етапи роботи програми

1) Запис вхідного звуку (команди, сказаної користувачем)
2) Розбиття звуку на фрейми та виділення ознак (MFCC-векторів)
3) Розпізнавання голосових команд
4) Аналіз отриманого тексту
5) Виконання голосової команди

Вхідні дані

Словник для розпізнавання

g2p_ukr.dict

```
1  виключи v yy k ll uj ch i
2  відкрий v i d k r y j
3  вісім v i ss ii mm
4  включи v k ll ju ch i
5  два d v aa
6  девять dd ee vv j ja tt
7  десять dd ee ss ja tt
8  закрий z a k r yy j
9  запусти z ay p u ss tt y
10 згорни z g oo r n y
11 Інтернет i nn tt i r nn je t
12 калькулятор k ay ll k u ll ja t ay r
13 комп'ютер k oo m pp j ju tt i r
14 мінус mm ii n u s
15 нуль n u ll
16 один oo dd yy n
17 ок oo k
```

Вхідні дані

JSGF-граматика

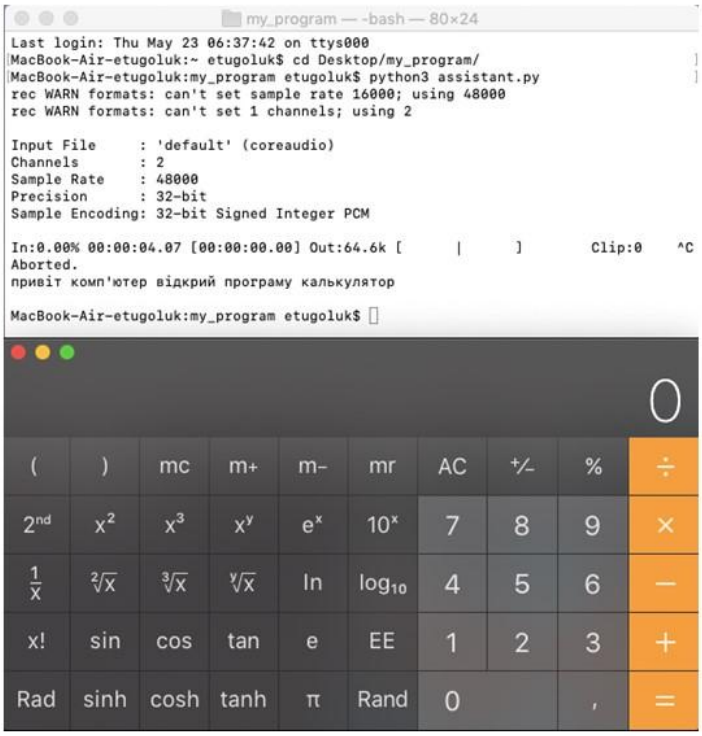
ukr.jsgf

```
#JSGF V1.0;

grammar ukr_calc;

<greeting> = (привіт|ок|окей)(комп'ютер);
<action> = (відкрий|включи|виключи|закрий|запусти|згорни|покажи|
сховай|розгорни);
<object> = (папку|програму);
<name> = (
Інтернет|Калькулятор|Пошта|Фото|Шахмати|Книги|Календар);
<expression> = <action> <object> <name>;
public <query> = <greeting> <expression>;
```

Приклади роботи програми



Приклади роботи програми

Результати роботи

- Програма розпізнає людські голоси різного тембру, статі та віку
- Найкращі результати програма показує при нормальному темпі мовлення та в безшумному середовищі
- При посторонньому шумі та у випадку багатоголосся отримуємо 60-80% коректно розпізнаних слів (може бути не розпізнано одне-два слова з речення)

Висновки

- Проведено аналіз методів розпізнавання мовлення
- Проведено аналіз існуючих програмних рішень
- Обґрунтовано використання обраних засобів розробки
- Створено програмний застосунок для розпізнавання української мови у вигляді системи голосового управління ОС

Дякую за увагу!

ДОДАТОК Б

g2p.py

```
#!/usr/bin/python
# -- coding: utf-8 --

import sys

softletters=set(u"яюїєі")
startsyl=set(u"#аяуюеєіїоиь-")
others = set(["#", "+", "-", u"ь", u""])

softhard_cons = {
    u"б" : u"b",
    u"в" : u"v",
    u"г" : u"g",
    u"Г" : u"g",
    u"д" : u"d",
    u"з" : u"z",
    u"к" : u"k",
    u"л" : u"l",
    u"м" : u"m",
    u"н" : u"n",
    u"п" : u"p",
    u"р" : u"r",
    u"с" : u"s",
    u"т" : u"t",
    u"ф" : u"f",
    u"х" : u"h"
}

other_cons = {
```

```
    u"ж" : u"zh",
    u"ц" : u"c",
    u"ч" : u"ch",
    u"ш" : u"sh",
    u"щ" : u"sch",
    u"й" : u"j"
}

vowels = {
    u"a" : u"a",
    u"я" : u"a",
    u"y" : u"u",
    u"ю" : u"u",
    u"o" : u"o",
    # u"ё" : u"o",
    # u"э" : u"e",
    u"e" : u"e",
    u"є" : u"e",
    # u"и" : u"i",
    # u"ы" : u"y",
    u"и" : u"y",
    u"i" : u"i",
    u"ї" : u"i",
}

def pallatize(phones):
        for i, phone in enumerate(phones[:-1]):
                if phone[0] in softhard_cons:
                        if phones[i+1][0] in softletters:
                                phones[i] = (softhard_cons[phone[0]] + "j", 0)
                        else:
                                phones[i] = (softhard_cons[phone[0]], 0)
```

```
            if phone[0] in other_cons:
                phones[i] = (other_cons[phone[0]], 0)

def convert_vowels(phones):
    new_phones = []
    prev = ""
    for phone in phones:
        if prev in startsyl:
            if phone[0] in set(u"яюєї"):
                new_phones.append("j")
        if phone[0] in vowels:
            new_phones.append(vowels[phone[0]] + str(phone[1]))
        else:
            new_phones.append(phone[0])
        prev = phone[0]

    return new_phones

def convert(stressword):
    phones = ("#" + stressword + "#").decode('utf-8')

    stress_phones = []
    stress = 0
    for phone in phones:
        if phone == "+":
            stress = 1
        else:
            stress_phones.append((phone, stress))
            stress = 0

    pallatize(stress_phones)
    phones = convert_vowels(stress_phones)
```

```
        phones = [x for x in phones if x not in others]

        return " ".join(phones).encode("utf-8")

for line in open(sys.argv[1]):
        stressword = line.strip()
        print stressword.replace("+", ""), convert(stressword)
```

ukr.dict

виключи v yy k ll uj ch i
відкрий v i d k r y j
вісім v i ss ii mm
включи v k ll ju ch i
два d v aa
девять dd ee vv j ja tt
десять dd ee ss ja tt
закрий z a k r yy j
запусти z ay p u ss tt y
згорни z g oo r n y
Інтернет i nn tt i r nn je t
Календар k a l ee n d aa r
Калькулятор k ay ll k u ll ja t ay r
Книги k n y g y
комп'ютер k oo m pp j ju tt i r
мінус mm ii n u s
нуль n u ll
один oo dd yy n
ок oo k
окей oo k ee j
папку p aa p k u
плюс p ll ju s
покажи p oo k a zh yy

```
Пошта p oo sh t aa
привіт pp r y vv i t
програму p r oo g r aa m u
пять pp j ja t
розгорни r oo z g oo r n y
сім ss ii mm
сховай s h oo v aa j
Термінал t ee r mm i n aa l
три t r yy
Фото f oo t oo
чотири ch ay t yy rr y
Шахмати sh aa h m aa t y
шість sh ii ss tt
```

ukr.jsgf

```
#JSGF V1.0;

grammar ukr_calc;

<greeting> = (привіт|ок|окей)(комп'ютер);
<action> =
(відкрий|включи|виключи|закрий|запусти|згорни|покажи|сховай|розгорни);
<object> = (папку|програму);
<name> = (Інтернет|Калькулятор|Пошта|Фото|Шахмати|Книги|Календар);
<expression> = <action> <object> <name>;
public <query> = <greeting> <expression>;
```

assistant.py

```
#!/usr/bin/python

import os
import re
```

```
def open_function(flag):
        print (flag)
        if (re.search(r'Калькулятор',result) != None):
                os.system("open " + flag + " Calculator")
        if (re.search(r'Інтернет',result) != None):
                os.system("open " + flag + " Safari")
        if (re.search(r'Пошта',result) != None):
                os.system("open " + flag + " Mail")
        if (re.search(r'Фото',result) != None):
                os.system("open " + flag + " Photos")
        if (re.search(r'Шахмати',result) != None):
                os.system("open " + flag + " Chess")
        if (re.search(r'Книги',result) != None):
                os.system("open " + flag + " Books")
        if (re.search(r'Календар',result) != None):
                os.system("open " + flag + " Calendar")


def close_function():
        if (re.search(r'Калькулятор',result) != None):
                os.system("killall Calculator")
        if (re.search(r'Інтернет',result) != None):
                os.system("killall Safari")
        if (re.search(r'Пошта',result) != None):
                os.system("killall Mail")
        if (re.search(r'Фото',result) != None):
                os.system("killall Photos")
        if (re.search(r'Шахмати',result) != None):
                os.system("killall Chess")
        if (re.search(r'Книги',result) != None):
                os.system("killall Books")
        if (re.search(r'Календар',result) != None):
```

```
                os.system("killall Calendar")

record_file = "out/out.wav"
record = "rec -r 16k -e signed-integer -b 16 -c 1 " + record_file
os.system(record)

format_file = ".wav"
control_file = "wav_file"
samplerate = "16000"

dictionary = "ukr.dict"
grammar = "ukr.jsgf"
model = "acoustic_model"

out_dir = "out/"
log_file = out_dir + "output.log"
out_file = out_dir + "out.txt"

recognition = "pocketsphinx_batch" + \
                        " -adcin yes " + \
                        " -cepdir " + out_dir + \
                        " -cepext " + format_file + \
                        " -ctl " + control_file + \
                        " -hmm " + model + \
                        " -jsgf " + grammar + \
                        " -dict " + dictionary + \
                        " -hyp " + out_file + \
                        " -samprate " + samplerate + \
                        " -logfn " + log_file
os.system(recognition)

f = open(out_file, "r")
```

```
result = re.sub(r' \(out.*', '', f.readline())
print (result)

flag = ""
if (re.search(r'програму',result) != None):
        flag = "-a"

if (re.search(r'(відкрий|включи|запусти|покажи|розгорни)',result) != None):
        open_function(flag)
if (re.search(r'(закрий|виключи|згорни|сховай)',result) != None):
        close_function()

f.close()
```

mfcc.py

```
import librosa
from librosa.feature import mfcc
import sklearn.preprocessing

class FeauturesCalculator:
    """
    Opens file, calculates features and returns them
    Attributes
    ----------
    _res_type : str
        parameter for librosa.load
        defines resample type
    _nfft : int
    _nmfcc : int
    _hop_length : int
        info for calculating mfcc (for librosa.feature.mfcc)
    _scale : bool
```

```
            scale mfcc or not
        """

        def __init__(self, nfft, nmfcc, hop_length=512, res_type='scipy', scale=True):
            self._res_type = res_type

            self._nfft = nfft
            self._nmfcc = nmfcc
            self._hop_length = hop_length

            self._scale = scale

        def getFeaturesFromWAV(self, filename):
            audio, sampling_freq = librosa.load(filename, sr=None,
res_type=self._res_type)

            features = librosa.feature.mfcc(
                audio, sampling_freq, n_mfcc=self._nmfcc, n_fft=self._nfft,
hop_length=self._hop_length)

            if self._scale:
                features = sklearn.preprocessing.scale(features)

            return features.T
```

model.py

```
import os
from hmmlearn import hmm
from sklearn.externals import joblib
import numpy as np
import warnings
warnings.simplefilter('ignore', DeprecationWarning)
```

```
class HMMParams:
    """
    Defines significant params for HMM
    """

    def __init__(self, n_components=4, cov_type='diag', n_iter=1000, tol=1e-4):
        self.n_components = n_components
        self.cov_type = cov_type
        self.n_iter = n_iter
        self.tol = tol


class HMMTrainer:
    """
    Wrapper for GaussianHMM
    """

    def __init__(self, hmmParams=HMMParams()):

        self._hmm =
hmm.GaussianHMM(n_components=hmmParams.n_components,
                                covariance_type=hmmParams.cov_type,
n_iter=hmmParams.n_iter, tol=hmmParams.tol)

    def train(self, X):
        np.seterr(all='ignore')
        self._hmm.fit(X)

    def get_score(self, input_data):
        return self._hmm.score(input_data)
```

```
def get_score_samples(self, input_data):
    return self._hmm.score_samples(input_data)

def get_monitorInfo(self):
    return self._hmm.monitor_

def save(self, folder_name, class_name, debug_mode=False):
    """
    folder_name : str
        folder which contains output
    class_name : str
        name of output file
    debug_mode : bool
        prints debug info is true
    """

    if folder_name[-1] != '/':
        folder_name += '/'

    filename = folder_name + class_name + '.pkl'

    if debug_mode:
        print('Start saving to ' + filename)

    joblib.dump(self._hmm, filename)

    if debug_mode:
        print('Saving completed ' + filename)

def load(self, folder_name, class_name, debug_mode=False):
    """
    folder_name : str
```

```
            folder which contains output
        class_name : str
            name of output file
        debug_mode : bool
            prints debug info is true
        """

        for filename in [x for x in os.listdir(folder_name) if (x.endswith('.pkl') and
(class_name in x))]:
            filepath = os.path.join(folder_name, filename)
            if debug_mode:
                print('Start loading  ' + filename)

            self._hmm = joblib.load(filepath)

            if debug_mode:
                print('Loading completed ' + filename)
```